

《面向对象设计与构造》

Lec04第一单元总结

OO课程组2024

吴际

北京航空航天大学计算机学院

提纲

- 单元教学目标的设置
- 单元训练的设计分析
- 作业和实验的效果分析
- 代码风格
- 博客作业

本单元教学目标设置

- 建立面向对象程序的认识
 - 面向对象程序的框架
 - 三个关键类：输入处理、主控、核心数据管理
- 认识对象的结构特征
 - 数据和行为两个维度
 - 序列、组合层次、递归层次、抽象层次
 - 迭代情况下的结构变化
- 理解和掌握层次化设计
 - 按照数据/行为建立抽象层次
 - 对多层次对象进行归一化管理
- 训练设计
 - 递进式的三次作业
 - 两次在线实验

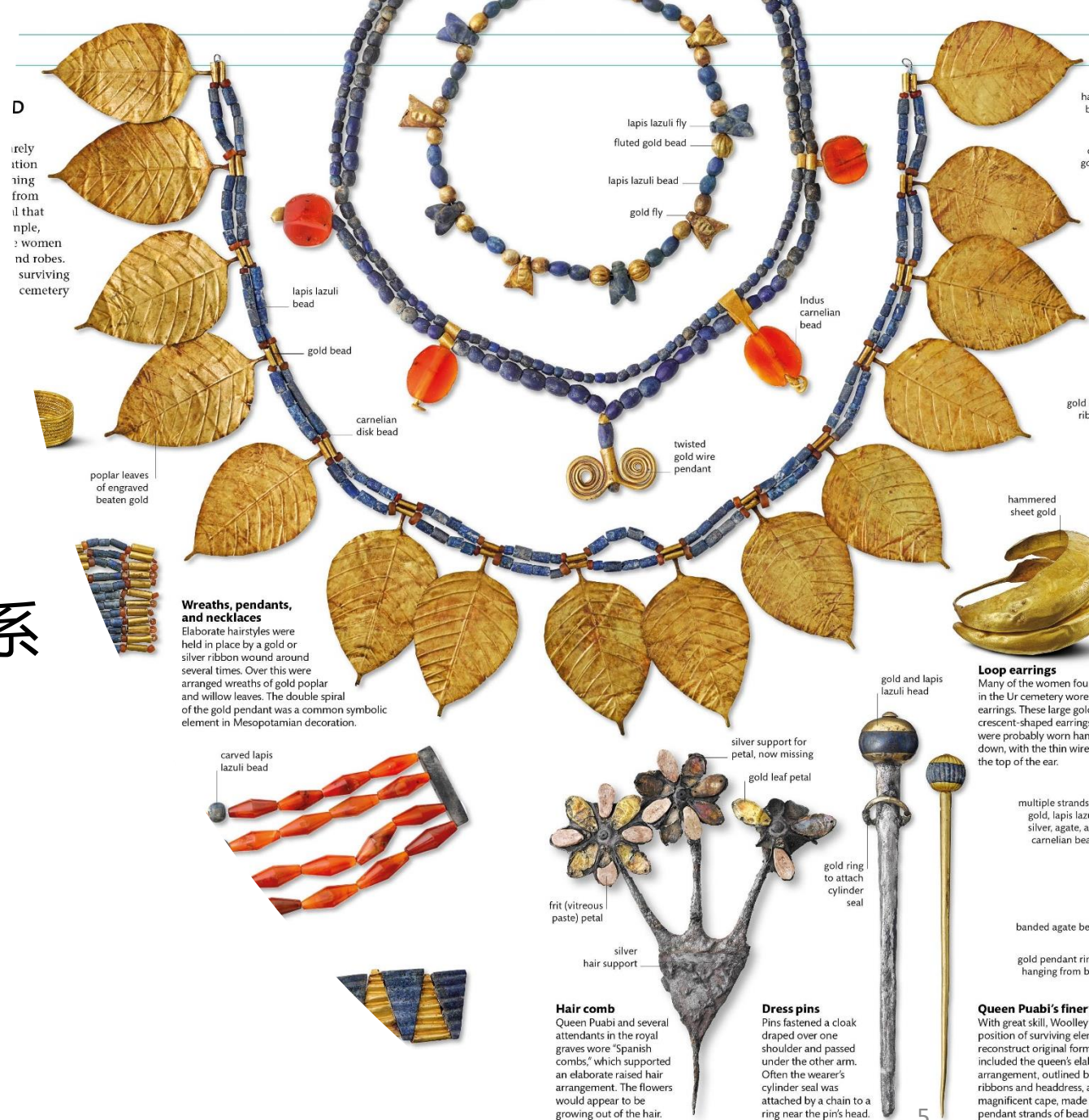
面向对象是一种思维方法

- 对象是具体个体，各自维护自己的状态
 - 对象具有自治性
 - 如果对象可以在不同的主机上执行，且还能互相交互如何？
- 同一个类所实例化对象，状态管理能力相同，但却是不同个体
 - 独立存在，状态差异
 - 一个对象出错，不意味其他对象必然出错
- 所谓“面向”，即把对象(类)作为单位来规划程序的结构和行为
 - 数据管理行为
 - 用户交互行为
 - 计算控制行为

对象层次→协同行为

面向对象是一种思维方法

- 三个基本问题
 - 如何管理对象
 - 如何建立对象之间的层次关系
 - 如何利用层次关系



面向对象是一种思维方法

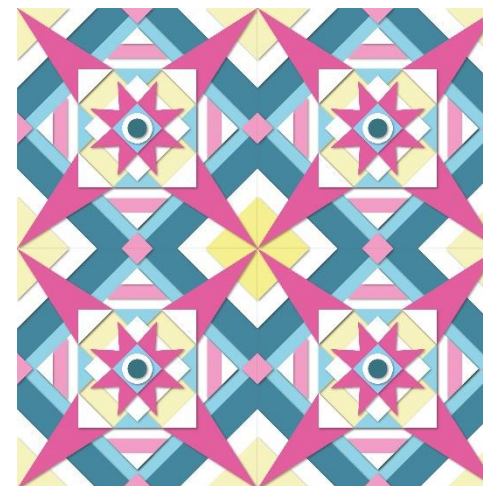
- 对象管理
 - 谁来管
 - 内部：自己构造，自己管理
 - 外部：传递工作内容(外→内，内→外)
 - 如何管
 - 个体枚举
 - 静态数组
 - 可伸缩容器
 - 可Hash快速访问容器（特征化）
- 如何用
 - 存储数据
 - 拷贝“外借”（深浅拷贝问题）
 - 层次代理



对象的创建与管理

- 本单元训练涉及多种类型的对象
 - 表达式、项、因子
 - 指数函数、自定义函数、函数调用、求导
- 组合规则更是复杂，且可以递归
- 解析输入并构造对象是个重要问题
 - 字符串解析：分散 vs 统一
 - 对象构造：分散 vs 统一
- 可以使用设计模式来重构

A pattern is a **regularity** in the world, man-made design, or abstract ideas. As such, the elements of a pattern **repeat** in a **predictable** manner. --wikipedia



Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

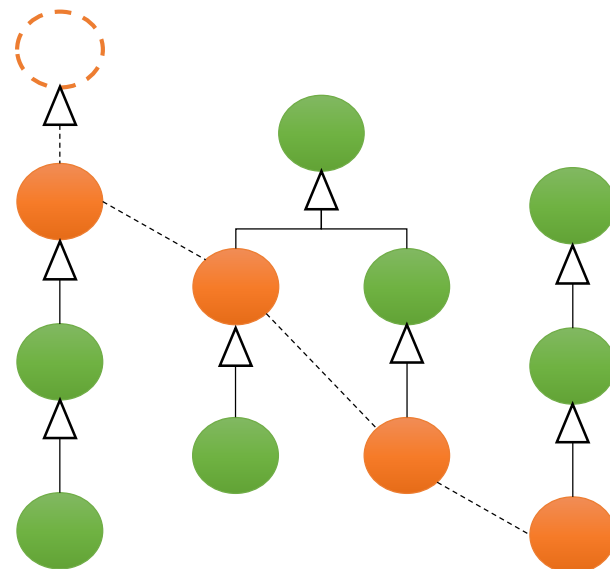
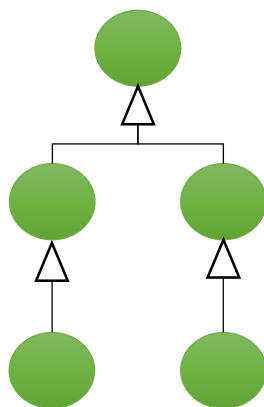
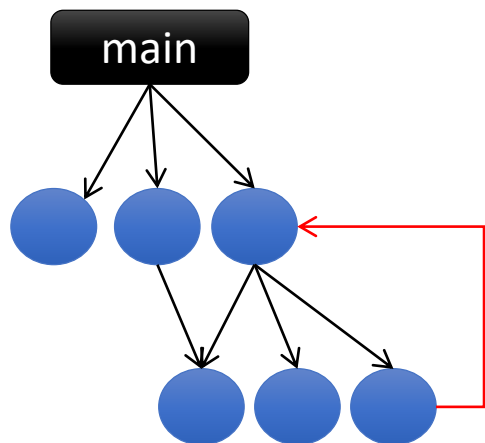


ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

对象构造模式
Singleton Pattern
Factory Pattern
Prototype Pattern
Builder Pattern
Object Pool Pattern

面向对象是一种思维方法

- 建立层次关系
 - 按照part-of关系形成的层次（组合层次）
 - 存在递归组合层次
 - 按照kind-of关系形成的层次（继承层次）
 - 按照can-do关系形成的层次（接口实现层次）
 - 三种层次关系可以叠加



面向对象是一种思维方法

- 层次关系管理
 - “意中人”在哪里
 - 从容器中找到
 - 对象引用带着**面纱**
 - 归一化管理
 - 通过上层抽象来无差别的引用和访问
 - 良好设计的结果
 - 协作是硬道理
 - 水平协同、跨层协同
 - 模式化(赋之以角色): Factory、Observer(publish/subscribe)、Visitor...



几乎大部分设计模式都会使用继承和接口来建立层次

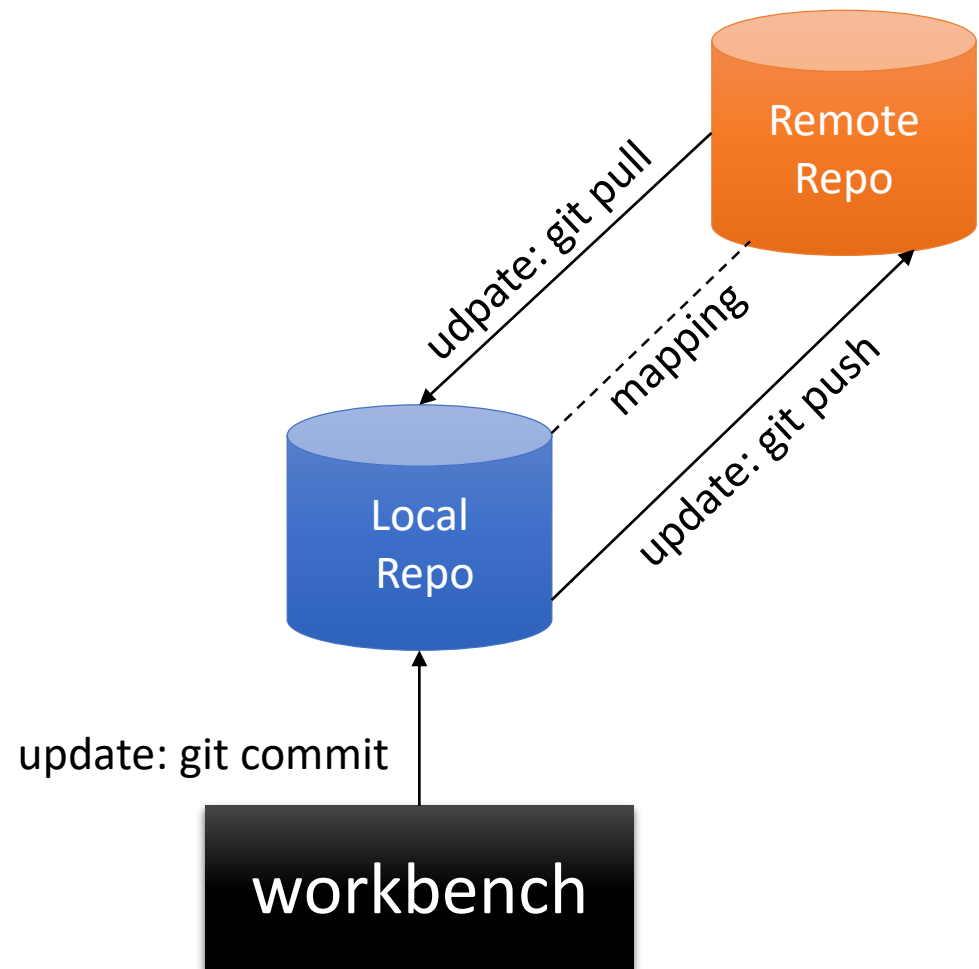
本单元作业的设计意图

第一次作业的设计目标

- 类的层次化设计：如何针对目标应用情景建类？
 - 多种类型的对象：表达式、项、因子（常量因子、变量因子、表达式因子）
 - 充分利用Java提供的容器方便管理对象
- 分工协作的行为设计
 - 输入处理
 - 结构化解析和信息存储
 - 处理展开与化简：乘法与合并同类项
- 可扩展性
 - 怎样的设计可以更好地方便后续可能的迭代需求？
 - 怎样的设计可以在迭代预测失误时减少修改的代价？
- 建立程序的鲁棒性概念
 - 输入处理的鲁棒性：递归下降法
 - 数据管理的鲁棒性：超大数，未知对象数量(非定长容器)
 - 计算的鲁棒性：中间过程的数据存储，消除冗余对象防止性能过慢(0值项)

本单元作业的设计意图

- 第一次作业的训练目标
 - 温习gitlab的使用
 - 什么是仓库?
 - git commit, git push, git pull在干啥?
 - 温习代码风格检查
 - 为什么强调代码风格?
 - 是独门暗器吗?
 - 重视和逐步掌握测试方法与技巧
 - 按照输入结构的测试设计
 - 按照文法定义构造生成器
 - 中测的合理玩法



本单元作业的设计意图

- 第二次作业的设计目标
 - 迭代设计
 - 增加函数与函数调用
 - 函数因子
 - 预定义函数与自定义函数
 - 指数函数
 - 函数定义与函数调用
 - 递归结构更加复杂
 - 表达式括号的多层嵌套
 - 函数因子→函数表达式
 - 增加行为类别
 - 展开：乘法、去括号
 - 合并：同类项(如何判断指数函数和其同类项？)
 - 替换：从字符串级别上做形式替换还是在逻辑结构上替换，可能对原输入造成的破坏？

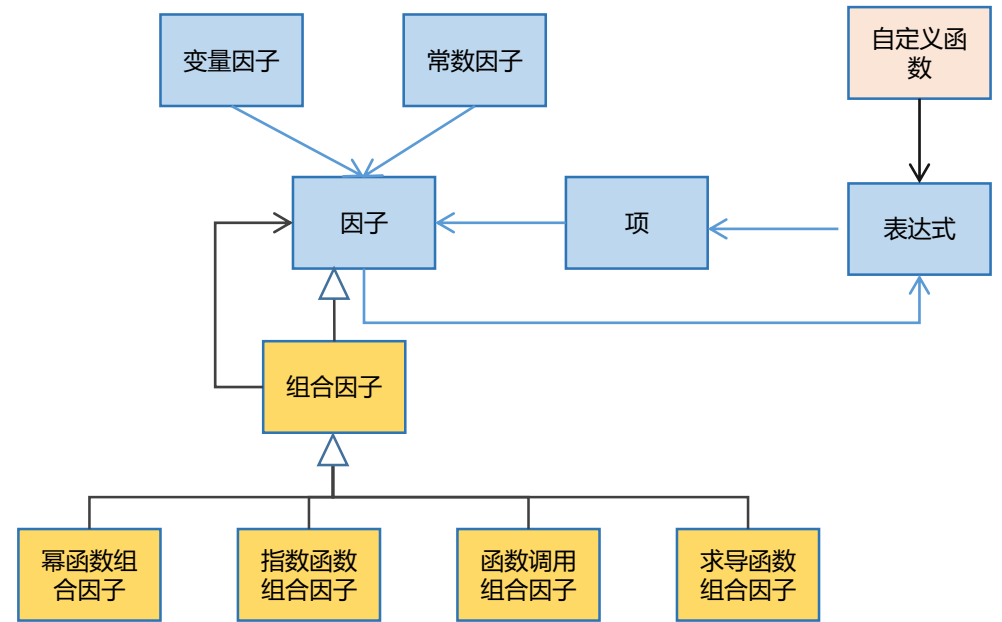
本单元作业的设计意图

- 第二次作业的训练目标

- bug修复: (pass the failed test case(s)) && !(fail any passed test case(s))
- 代码重构
 - 建立组合层次关系 (递归结构的表示, exp中的因子)
 - 在层次间建立协同行为 (避免 “一竿子插到底” 的程序行为)
- 如何保证输出符合文法定义?
 - 输出的处理逻辑
 - 如何自动化检查输出是否符合文法?
 - 适度优化: 不见得能找到全局最优的结果

本单元作业的设计意图

- 第三次作业的设计目标
 - 在多种类型之间建立抽象层次
 - 有几种表达式/项/因子?
 - 进行归一化处理 (继承与接口)
 - 增加新的组合规则
 - **求导因子、作用于任意表达式**
 - 组合规则可递归应用



- 因子的行为抽象
 - 乘展开
 - 代入展开
 - 合并同类项
 - 解析构造或生成
- 因子的数据抽象(函数化)
 - 系数、幂
 - 实参列表
 - 函数表达式
- 多种函数因子
 - 幂函数
 - 指数函数
 - 函数调用
 - 求导函数

本单元作业的设计意图

- 第三次作业的训练目标
 - 掌握和应用继承、接口和多态机制
 - 面向应用的业务需求
 - 以统一的架构来整合三次作业的功能
 - 要以迭代演化的视角来看待三次作业
 - 强调架构的意义
 - 保持程序结构的简洁
 - 灵活应对需求变化
 - 如何基于黑盒测试来定位程序Bug
 - 复杂的程序内部状态导致失败的输入和bug位置之间的逻辑联系也复杂了
 - 在确保仍然能够重现失败的情况下，对输入进行化简
 - 程序中增加内部状态的输出观察
 - 读取信息与修改信息分离: toString方法的无副作用要求
 - 程序错误的可复现性：顺序容器与非顺序容器(arrayList or hashSet?)

本单元作业的设计意图

- 深入理解程序运行机理的重要性
 - 还记得上节课的addAll么?
 - addAll的实现机制就是调用add

```
class MyHashSet<T> extends HashSet<T> {  
    private int addCount;  
    public MyHashSet() {super(); addCount=0;}  
    public boolean add(T a) {  
        addCount++;  
        return super.add(a);  
    }  
    public boolean addAll(Collection<? extends T> c){  
        addCount += c.size();  
        return super.addAll(c);  
    }  
    public int getAddCount() {  
        return addCount;  
    }  
}
```

```
MyHashSet<String> mhs = new MyHashSet<>();  
mhs.addAll(Arrays.asList("D", "E", "F"));
```

```
@Override  
public String toString() {  
    int cnt = 0;  
    for (Factor factor : factors) {  
        if (factor.isNeg()) {  
            cnt++;  
            factor.setPos();  
        }  
    }  
    StringBuilder sb = new StringBuilder();  
    if (cnt % 2 == 0) {  
        sb.append("-");  
    }  
}
```

很好理解吧，比如查询一个对象写为字符串的形式，以字符串作为返回值的方法 `toString()` 应满足对象保持不变，无论连续调用多少次都应返回同一结果，对象自身保持不变，否则我们就将欣赏到调试闹鬼奇观。

<http://oo.buaa.edu.cn/assignment/505/discussion/1516>



数据结构设计还是层次化设计？

- 不少同学通过容器嵌套来表示数据层次关系
 - 如List<List<Factor>>
 - 导致在遍历访问、展开和合并时非常复杂
- 首先建立数据抽象层次
 - 基础：函数（系数、幂）
 - 层次依据：多种不同的函数
- 然后建立基于函数的组合管理层次
 - 线性组合、乘积组合、嵌套组合
 - 组合结构依赖于**抽象**，而不是具体！ → **每一种组合的结果仍然是一种函数！**

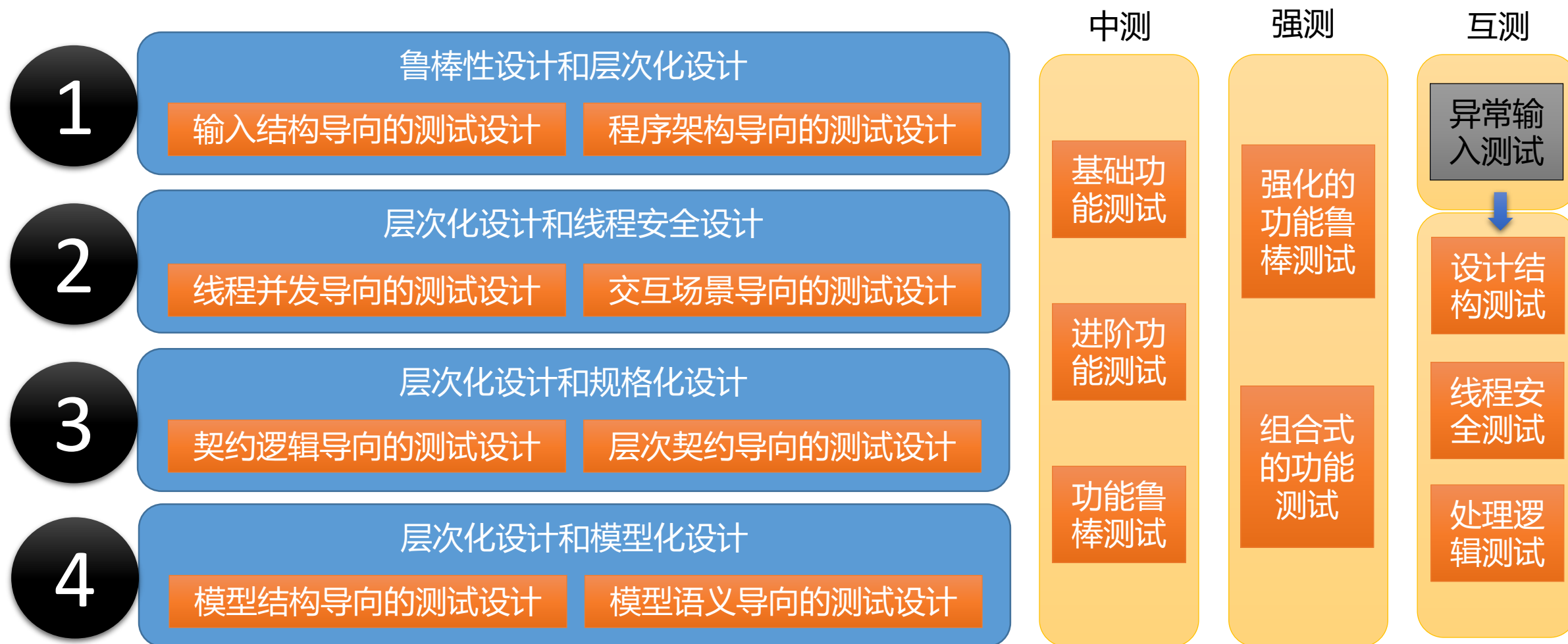
本单元作业的测试实践

- 黑盒测试关注功能性和鲁棒性
 - 依据需求/指导书来设计测试用例
 - 正常测试用例~异常测试用例
- 输入的组合
 - 具有明确的pattern和组合性
 - 表示结构：线性序列结构
 - 逻辑结构：带递归的层次结构
- 按照指导书的文法递归向下来生成数据实例
 - 每个**对象**都要出现
 - 每种组合规则都要出现
 - 每种符号模式都要出现
 - 每种系数模式都要出现

本单元作业的测试实践

- 测试是一面镜子
 - 反射镜：测试别人，发现自己的问题
 - 放大镜：通过测试放大程序中的局部问题，直至架构问题
 - 望远镜：结合需求迭代预测可能的脆弱点，提前进行重构或固化
- 三次作业之间具有很强的递进性
 - 测试也是如此
 - 后一次作业兼容前一次作业的功能和测试用例
 - 便于开展回归测试
 - 同学们应思考在这种迭代开发模式下，如何管理好测试的迭代？

课程作业的设计及其测试路线图



不能忽视先导课的价值

- 测评流程的熟悉：
 - git提交作业的流程：配置环境，熟悉命令
 - 中测-强测的流程：习惯一次作业的节奏
 - 网站的使用：如何查看报错信息
- Java语言的熟悉：多次作业和针对性训练
 - 先导课从简单到搭建一个复杂的系统<--->U1第一次作业直接建立一个复杂的系统
 - IDEA的熟悉和使用
 - 代码风格检查：规范命名和规范格式
 - java的基础语法
 - 如何建类？如何运行？如何定义方法和属性？
 - java的容器使用：不同容器的选择与比较？<----> 我该用什么容器管理语法树？
 - 栈，堆，内存模型<--->栈空间和堆空间不足的报错是怎么回事？
 - 常见bug：深浅拷贝？判等原则？<-----> 怎样化简？

不能忽视先导课的价值

- 面向对象思想的熟悉：

- 不同种类的瓶子和武器<--->不同种类的因子
- 封装不同方法的具体实现：多条命令处理<--->解析多种语法成分
- 输入处理类与业务执行类分离<--->lexer、parser
- 多个类的组织与关联
- 设计模式和其对架构设计的启发

- 程序实践技能的掌握：

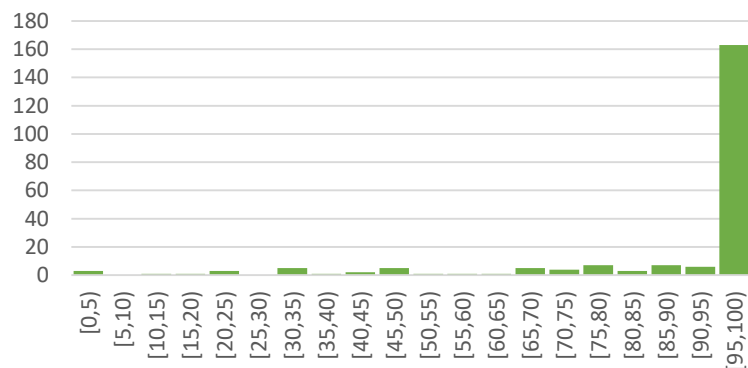
- debug与调试能力
- 测试基本思想的掌握与理解
- 如何应对先导课作业的迭代开发？重构中培养程序的可扩展性素养
- 博客实践：学会总结和分享你的学习成果

先导课对U1及整个OO课程的顺利开展有很大的帮助！

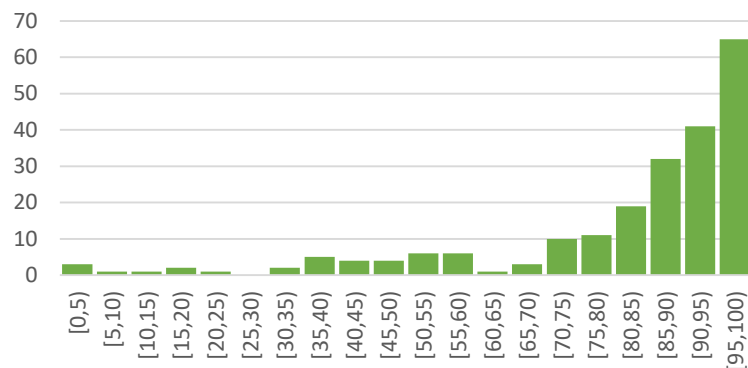
先导课中强调的测试能力还需要进一步体会提高

不能忽视先导课的价值

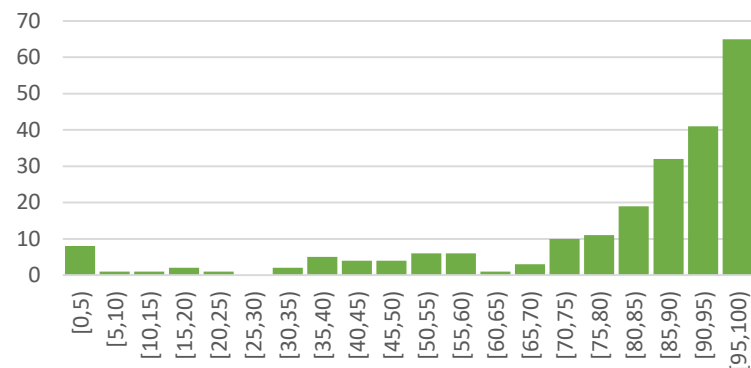
HW1中参加过先导课训练的强猜测成绩分布



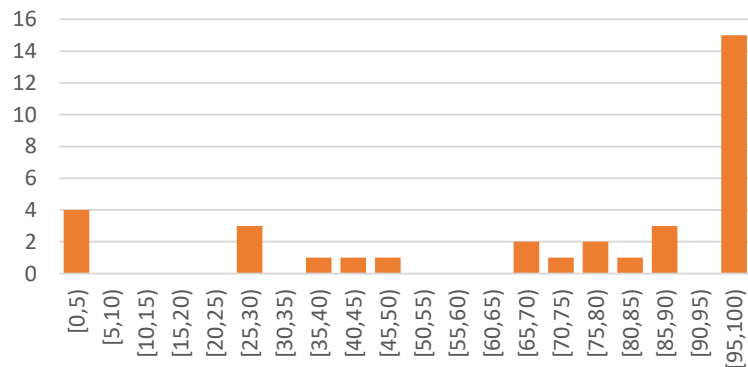
HW2中参加过先导课训练的强猜测成绩分布



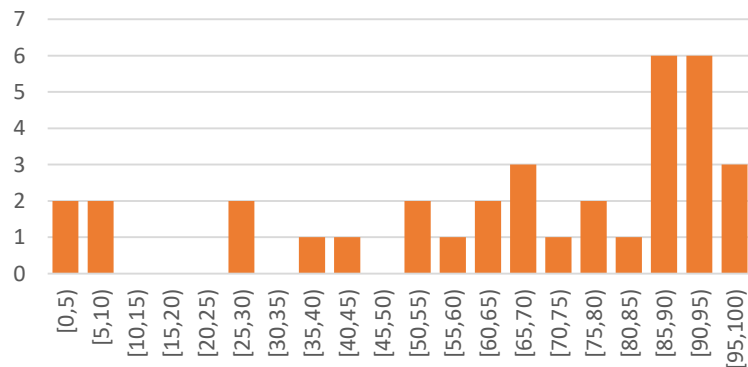
HW3中参加过先导课训练的强猜测成绩分布



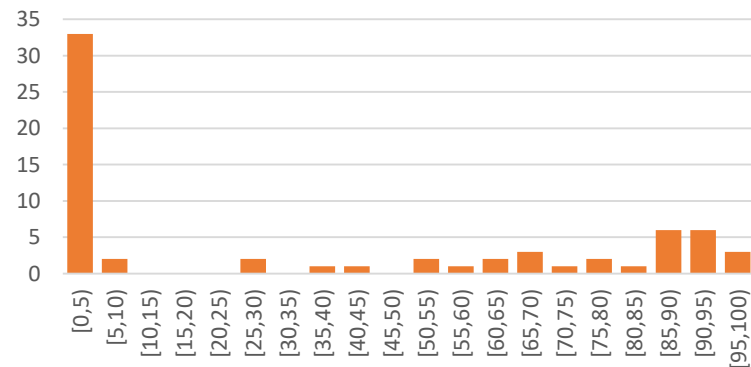
HW1中未参加先导课训练的强猜测成绩分布



HW2中未参加先导课训练的强猜测成绩分布



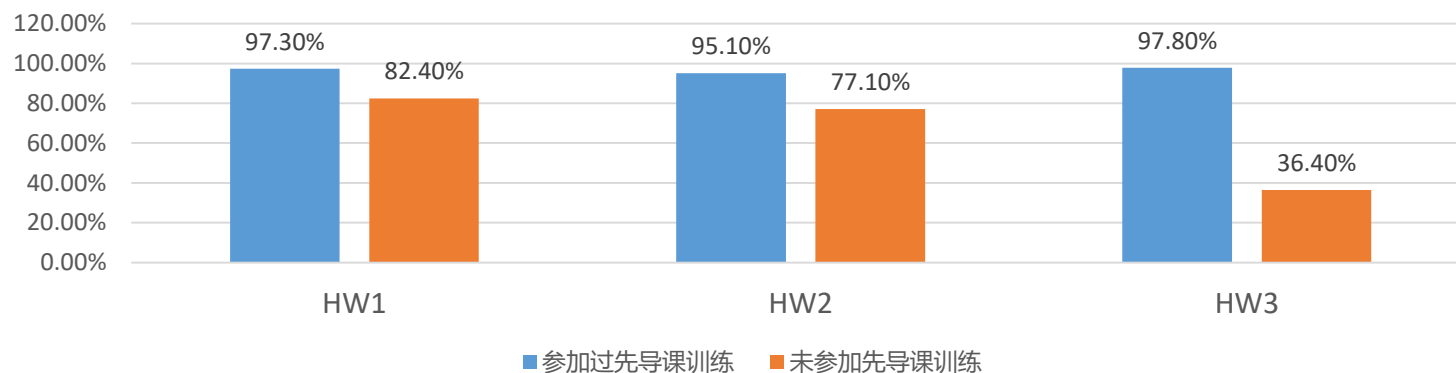
HW3中未参加先导课训练的强猜测成绩分布



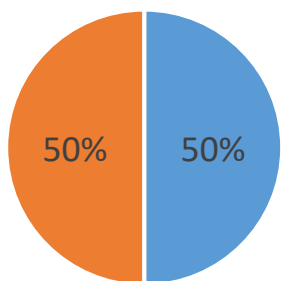
第三次作业中参加过先导课训练的学生强测成绩**明显高于**未参加的学生

不能忽视先导课的价值

三次作业两类学生有效作业占比

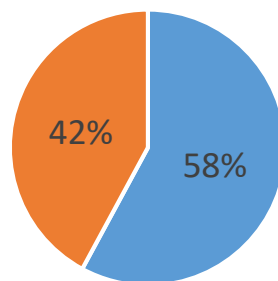


HW1中无效作业中两类同学占比



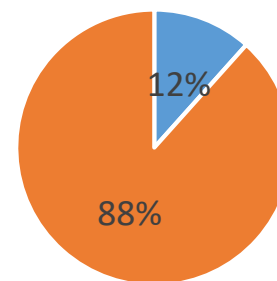
■ 参加过先导课训练: 6人 ■ 未参加先导课训练: 6人

HW2中无效作业中两类同学占比



■ 参加过先导课训练: 11人 ■ 未参加先导课训练: 8人

HW3中无效作业中两类同学占比

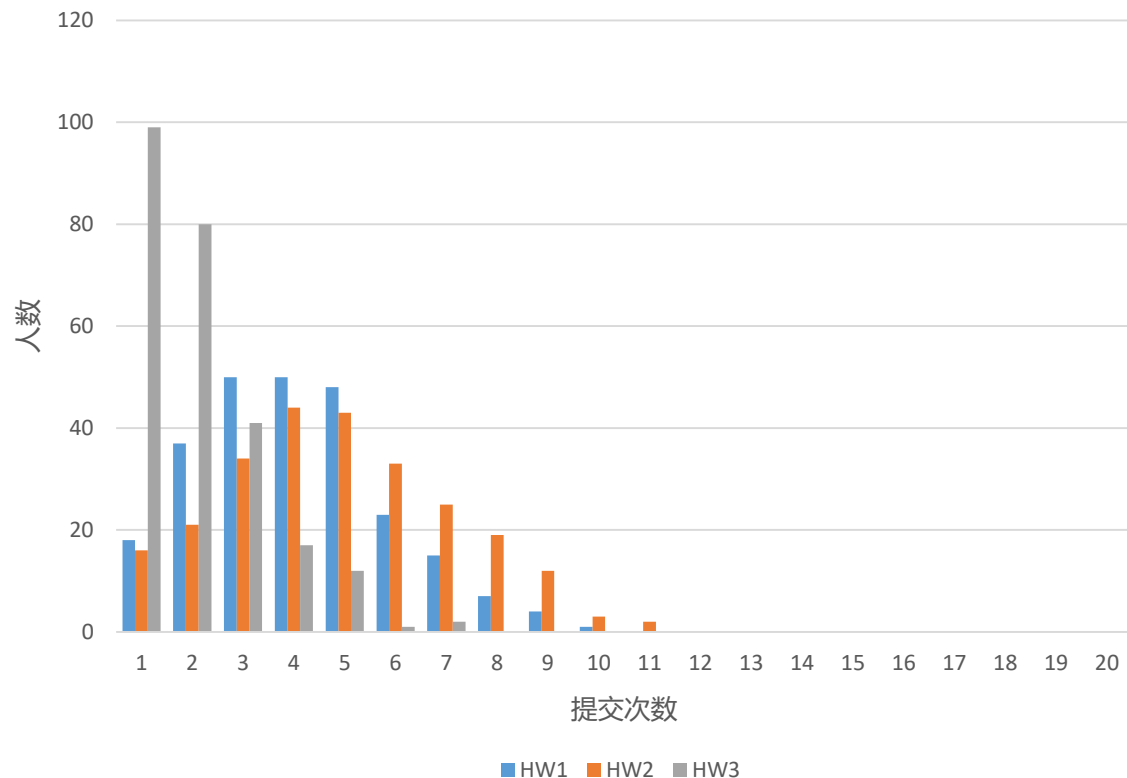


■ 参加过先导课训练: 5人 ■ 未参加先导课训练: 38人

第三次作业中对比非常明显，随着难度的加大，基础的重要性越来越显现

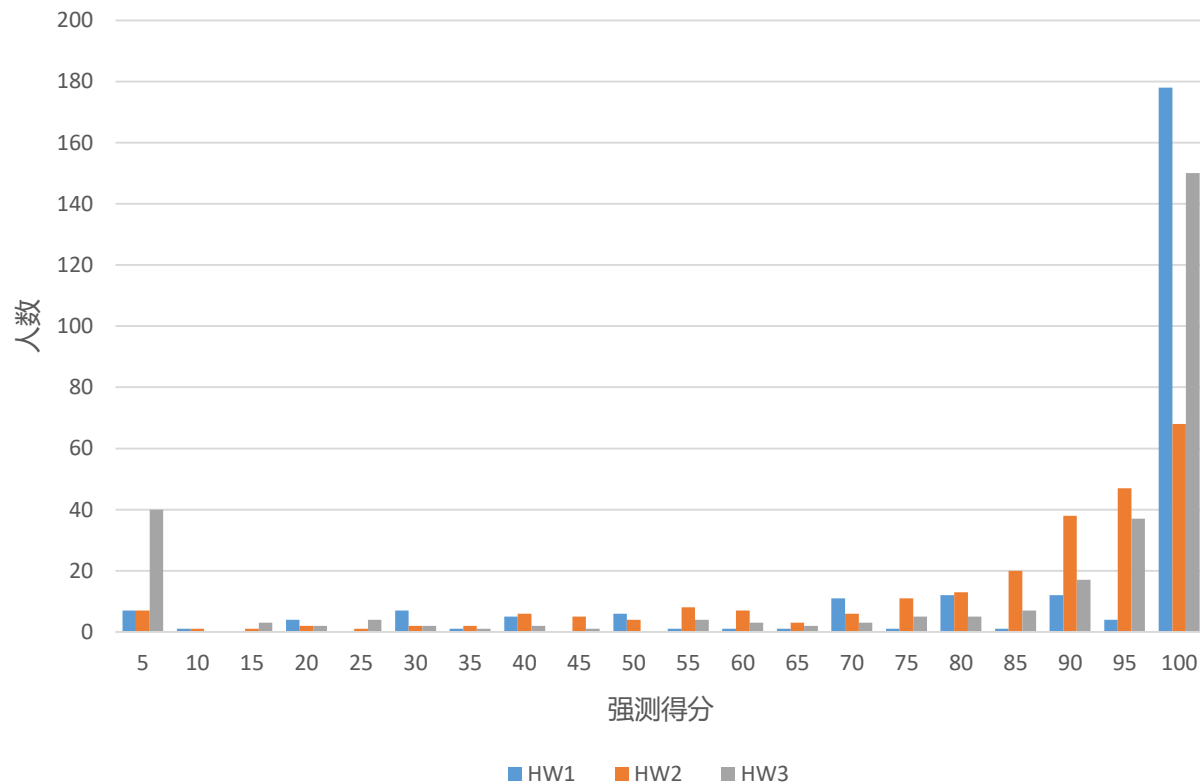
三次作业数据分析：公测及强测情况

提交次数分布



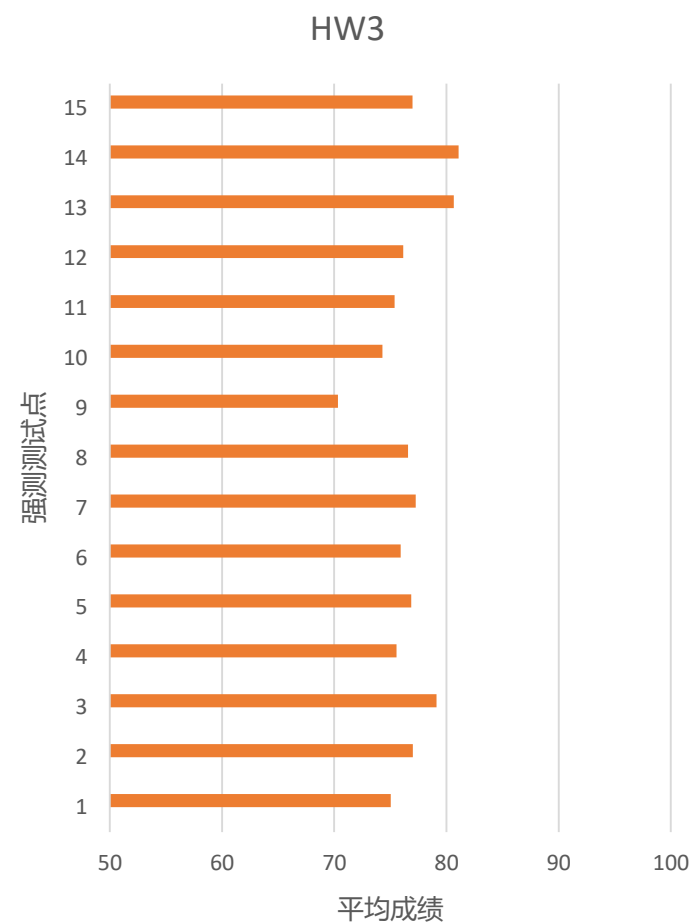
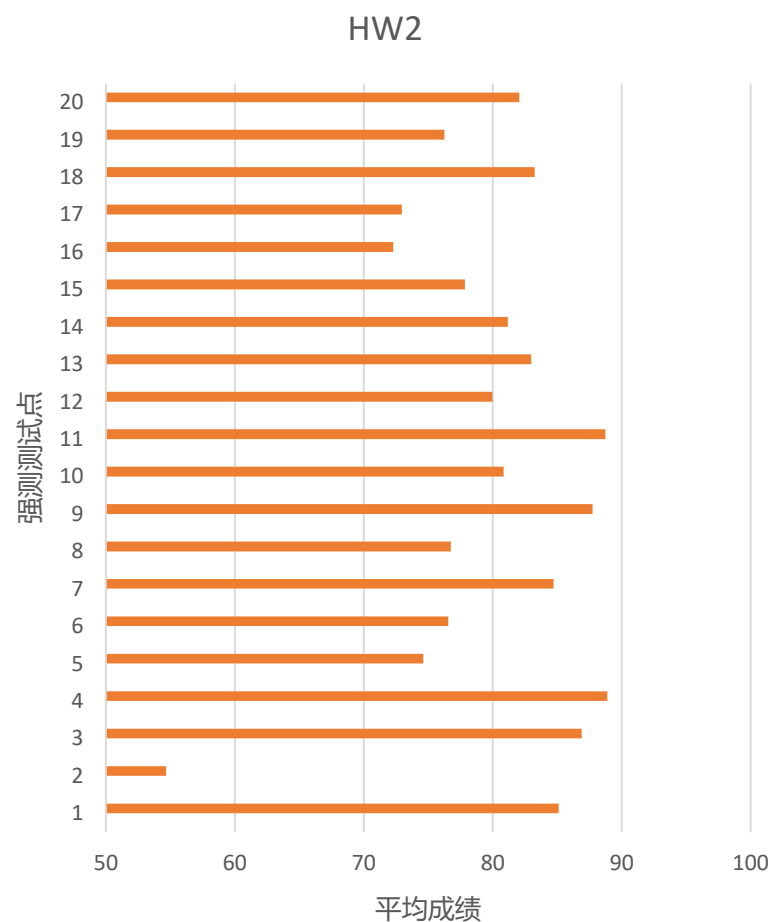
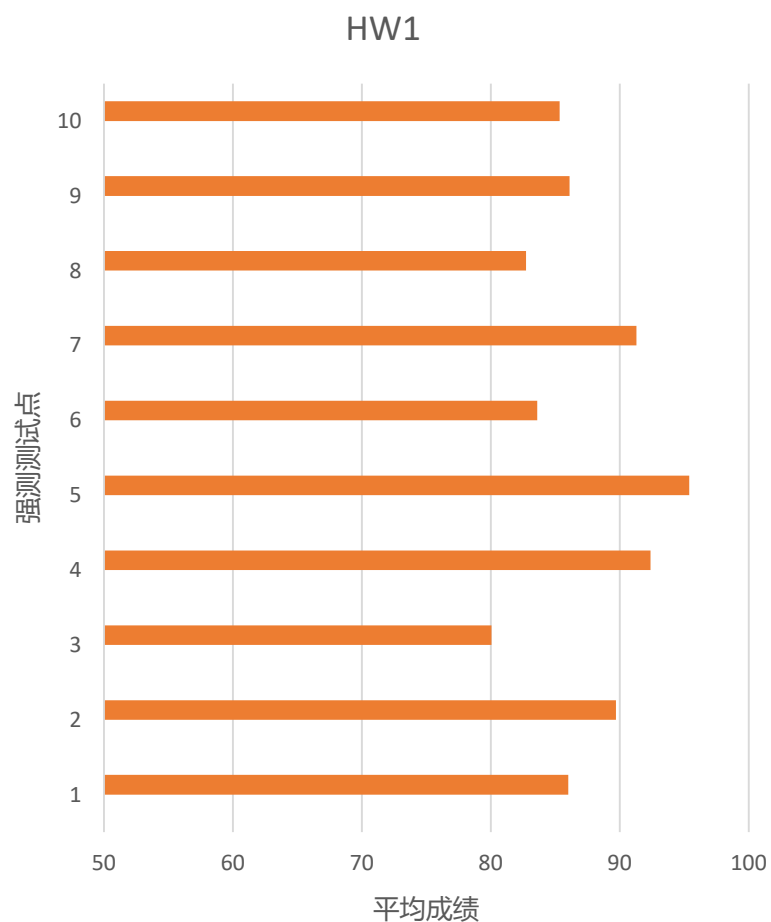
公测通过提交次数逐渐减少
逐渐掌握知识点，进步迅速

整体强测成绩分布



强测成绩占比稳定，**逐次平滑迭代**
在**架构设计**加强引导，作业质量有所提升

三次作业数据分析：数据点得分情况

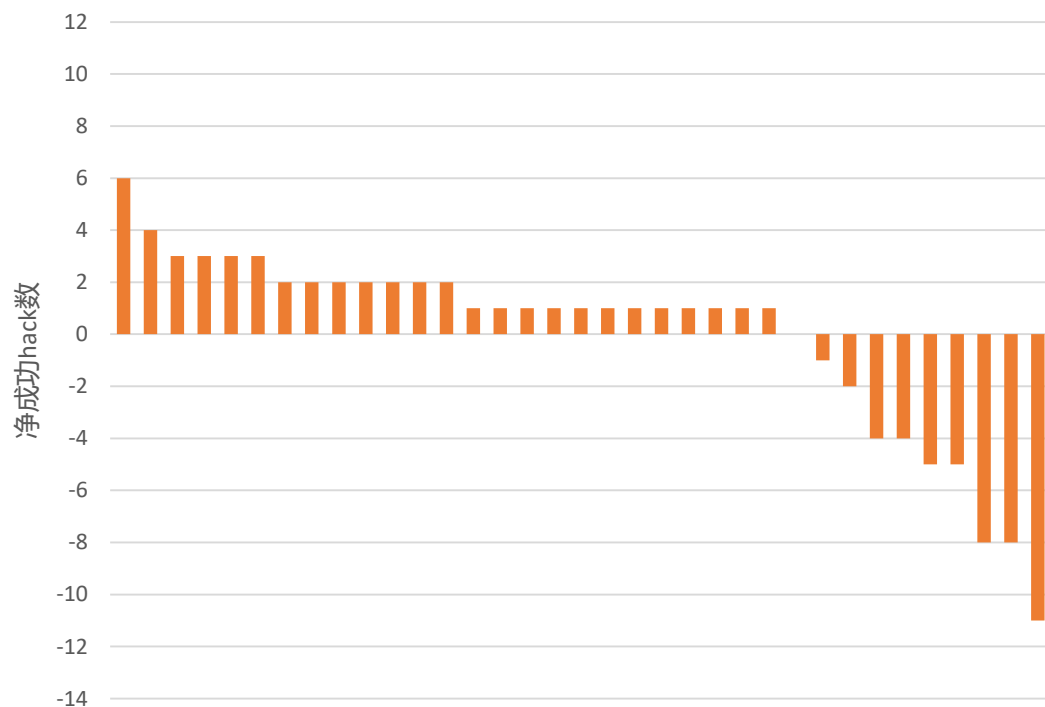


三次作业数据分析：数据点得分情况

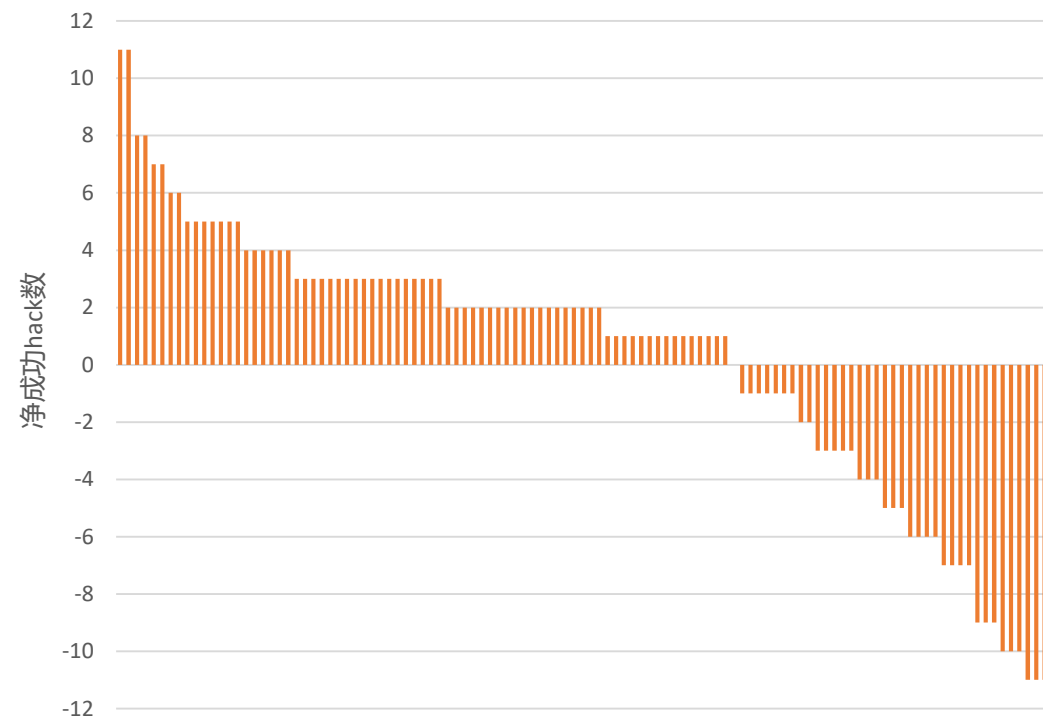
- HW1成绩偏低的强测数据点：**解析与大数处理**
 - 指数带符号
 - 合并同类项优化（按照指数合并系数，具有较长的数据与系数）
- HW2成绩偏低的强测数据点：**组合模式**
 - 多种实参因子、自定义函数调用带指数
 - 自定义函数定义形参顺序不是 x, y, z
 - 多次调用同一个自定义函数
- HW3成绩偏低的强测数据点：**组合嵌套**
 - 指数函数嵌套调用自定义函数
 - 自定义函数函数嵌套调用
 - 自定义函数因子嵌套求导

互测得失分分析

HW1互测hack情况



HW2互测hack情况



任何一个Level的room，成功hack和成功被hack数处于**相对的平衡状态**

互测分析：hack成功率

作业	房间类型	总人数	hack 次数	平均 hack 次数	成功 hack 次数	hack 成功率
HW1	A	137	4915	36	52	0.0106
	B	57	2675	47	61	0.0228
	C	40	2265	56	247	0.1091
HW2	A	162	6324	39	339	0.0536
	B	40	1292	32	125	0.0967
	C	16	910	57	56	0.0615

各级别互测意愿几乎相近，级别越高成功率越低，级别越低**互测补偿价值越高**

实验1效果分析

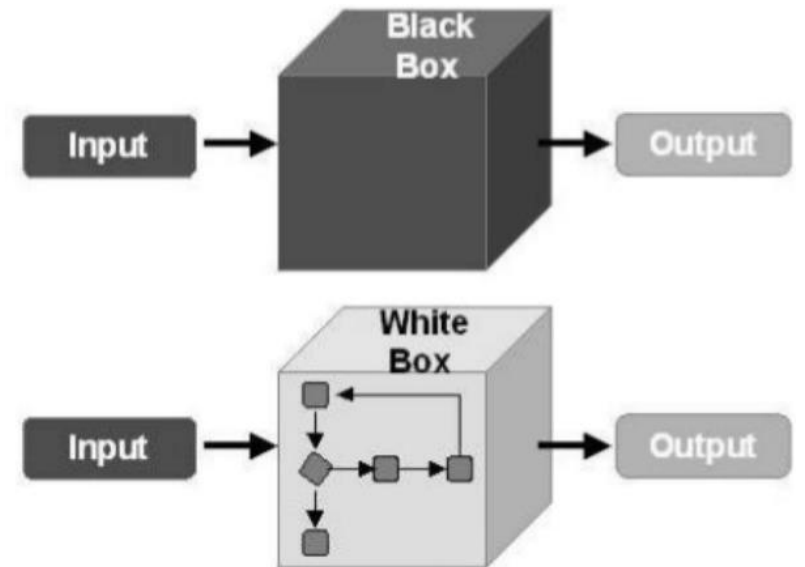
- 按时提交比例**86.9%**
 - 但依然还有少部分同学git操作不熟练
- 任务：表达式解析与计算
 - 训练目标：提供一个将表达式解析为二叉树的主干代码，理解代码逻辑并填充代码剩余部分——**可转化为第一次作业表达式的解析器**
 - 训练要点：Java 基础语法，对已给代码的解析逻辑的理解
 - 训练要点：理解对象的层次化管理并借鉴（树形结构）

实验2效果分析

- 按时提交比例**89.7%**
- 训练目标：基于给定的层次结构来增量实现求导方法，为作业项目提供指导
- 训练要点：
 - 理解已给代码在对象管理和处理方面的层次化架构
 - 理解求导规则会作用到哪些对象，能否归一化

如何分析和度量代码质量

- 黑盒分析
 - 功能测试 ✓ (中测、强测和互测)
 - 性能测试 ✓ (性能分)
- 白盒分析
 - 人工阅读、代码审查(Code Review) ✓ (互测)
 - 代码静态分析
 - 代码风格检查 ✓ (风格分)
 - 代码静态特征
 - 代码动态分析
 - 代码行为特征



测试的基本原则

- 独立视角，不能把自己限定成设计和开发人员，怎么用就怎么测
- 尽早介入，在需求确定时就开始，设计之初就思考怎么测试每个类
 - TDD：Test Driven Development
 - 规划输入的划分与组合
- 可靠追溯，一旦测试发现错误，需要追溯相应的代码和数据
- 确保复现，以规范方式复现问题
 - 逐步化简测试输入，在化简过程中获得了bug位置的理解
- 有序开展，测试输入的复杂度和规模由小及大
- 测不全是方法学上的缺陷，无法穷举所有可能
 - 要控制复杂度。软件模块越复杂，测不全带来的问题越严重

测试的重要性和局限性

- 是重要的质量保证手段
 - 内部质量决定外部质量
 - 软件质量是设计出来的
- 提高设计的可测性
 - 降低逻辑的复杂度
 - 提供中间状态的可观察性
 - 提供主动防御能力，提前发现数据问题
- 测不全问题始终存在
 - 存在潜在缺陷，核心程序需要进行严格验证（如机载、弹载和箭载系统等）
 - 验证手段：规格化+模型化
- 测试需要考虑用户的使用逻辑
 - 基于用户行为习惯的测试，互联网软件测试广泛采用，beta测试

单元测试

- 单元测试是非常有效的办法
 - 语句覆盖度100%
 - 分支覆盖度100%
 - 错误处理路径覆盖100%
 - 不只是功能，也要关注性能
 - 使用额定数据、异常数据、边界数据
 - 了解等价类的概念
- 可以尝试使用“Junit+执行脚本”手段来部署自动化的单元测试能力
- 单元测试是TDD的核心手段



静态检查

- 静态检查(不让计算机执行代码)也是一种重要的验证手段
 - 人们积累了不少代码错误模式
 - 通过检查代码是否出现某些错误模式来进行验证
 - 通过检查代码的逻辑来解决测试难以覆盖的问题
- 常见的检查要点
 - 调用是否对返回值进行了接收和检查
 - 类库及方法的使用是否符合要求
 - 容器访问的越界保护
 - 对象拷贝是否彻底
 - 数值计算可能溢出
 - 是否在循环体内对循环变量进行修改

代码静态特征：面向对象设计相关

- 类的继承深度反映抽象程度
 - Depth of Inheritance Tree(DIT)
 - 保持合适的度
- 不提供方法的类
 - 只定义少量属性，弱化为传统的数据结构定义
- 不管理数据的类
 - 如果只关心行为，应定义为接口
- 循环依赖的类
 - 多个类之间相互依赖，典型的职责划分不清（未形成层次结构）

代码静态特征：控制类和方法的长度

- 设计原则：高内聚、低耦合
- 绕不开的类
 - 几乎所有的类都与它交互→类代码行庞大
- 拥有巨型方法的类
 - 单个方法的规模远超其他方法规模之和
 - 为什么需要这样的方法？

代码静态特征：潜在的Bug区域

- 复杂方法：存在过多控制流结构以及嵌套使用
 - While, for, if, switch, do-while等多重嵌套使用
 - 控制流复杂，不易理解和调试
 - 复杂的逻辑难以理清边界，隐藏bug是必然的
- 复杂条件：条件嵌套使用
 - 与、或、非
 - 逻辑计算复杂
 - 难以准确定义真假分支对应的数据边界

开源静态分析工具DesigniteJava
(<https://github.com/tushartushar/DesigniteJava>)

解决办法

- 将复杂问题进行层次分解和抽象
 - 类数据层次
 - 类行为层次
- 类之间：有效协同，职责分派
- 类内部：行为解耦
 - 每个方法立足于类的属性数据来规划行为
 - 只做一件明确的事情
 - 方法代码行数 and 复杂度自然得到控制
 - 鼓励方法通过类所管理对象提供的行为来完成任务
 - 任务分派，实现解耦

为什么要探讨代码风格

- 一个基本定律：写代码容易，读代码难
 - 有一个游戏：醒来你发现自己被随机丢在某个城市的街景地图里，没有路名，没有地图，只有街景。你要自己找到机场，飞回家
- 在互测过程中，读到别人的代码是什么感受？
 - 结构清晰、易于理解的代码：心情愉悦，学习心态
 - 结构混乱、难以理解的代码：心情烦躁，发泄心态
- 你想成为哪个？

- 街景~局部代码
- 机场~预期想找的‘对象’
- 没有路名~看不懂代码中的命名
- 没有地图~看不懂代码中模块之间的逻辑关系

狭义的代码风格

- 程序员长期以来养成的一些编写代码的习惯
- 格式规范
 - 换行、缩进、长句断开.....
- 命名约定
 - 常量命名、变量命名、包命名、类和接口命名、方法命名.....
- 文档约定
 - 类和接口描述、方法描述.....
- 其他约定

广义的代码风格

- 坚持美观，灵活对待，符合编程的一般原则
 - 避免重复原则：一旦重复某个语句或概念，可以进行抽象
 - 抽象原则：与“避免重复”相关，一个功能只出现在一个位置
 - 简单原则：简单的代码占用资源少，漏洞少，易于修改
 - 避免创建不必要的代码：除非需要，否则不创建新功能
 - 尽可能做最简单的事：简化每一个类的功能，保持简单的路径
 - 别让我思考：代码要易于理解，不要让别人敬而远之
 - 开闭原则：可以基于你的代码进行拓展，但不能修改你的代码
 - 代码维护：本人和他人都能够容易维护
 - 最小惊讶原则：尊崇约束，减少给别人的惊喜或惊吓

代码风格的必要性

- 狭义的代码风格如同一身得体的打扮，能够给人留下第一印象
- 广义的代码风格体现能够写出“专业代码”的专业态度
- 腾讯、华为等各大企业都对代码风格进行了明文规定
 - 程序板式、注释、标识符命名等基本约定
 - 程序可读性、变量及结构体使用、可测性、可维护性等编程约定
 - 代码编辑、编译、审查等行为约定
 - 提供编码模板：可读性至上，遵循正确约定
- 专业能力的提高，伴随着代码风格的成熟
- 专业能力提升代码风格，代码风格体现专业能力

作业说明

- 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)、分析课程所介绍的共性问题
 - 在[csdn班级博客](#)上发布，注意发布的操作步骤
 - 鼓励同学们互相阅读和学习，并积极点评
 - (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（鼓励做更深入的度量分析）
 - 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
 - 画出自己作业类图，并自我点评优点和缺点
 - 注意1: **不要**使用工具“无脑”逆向生成类图
 - 注意2: **需要**配文字来解释每个类的设计考虑

作业说明

- (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
 - 对比分析**出现了bug的方法**和**未出现bug的方法**在**代码行**和**圈复杂度**上的差异
- (3)分析自己发现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（包括自测和互测）
- (4)架构设计体验
 - 结合三次作业的迭代介绍自己的架构如何逐步成型
- (5) 心得体会
 - 先导课带来的帮助
 - 其他相关的心得体会