



2022 年春季学期

计算学部《软件构造》课程

Lab 3 实验报告

姓名	李世轩
学号	120L022109
班号	2003007
电子邮件	1146887979@qq.com
手机号码	15234117960

目录

1 实验目标概述.....	1
2 实验环境配置.....	1
3 实验过程.....	1
3.1 待开发的三个应用场景.....	2
3.2 ADT 识别与设计.....	2
3.2.1 任务 1：投票类型 <code>VoteType</code>	2
3.2.2 任务 2：投票项 <code>VoteItem<C></code>	4
3.2.3 任务 3：选票 <code>Vote</code>	5
3.2.4 任务 4：投票活动 <code>Poll<C></code> 的测试.....	5
3.2.5 任务 5：投票活动 <code>Poll<C></code> 的实现类 <code>GeneralPollImpl</code>	6
3.2.6 任务 6：投票活动 <code>Poll<C></code> 的子类型.....	10
3.3 ADT 行为的设计与实现.....	11
3.3.1 任务 7：合法性检查.....	11
3.3.2 任务 8：采用 <code>Strategy</code> 设计模式实现灵活的计票规则.....	12
3.3.3 任务 9：采用 <code>Strategy</code> 设计模式实现灵活的遴选规则.....	13
3.3.4 任务 10：处理匿名和实名投票.....	13
3.3.5 任务 11：采用 <code>Visitor</code> 设计模式实现功能扩展.....	16
3.3.6 任务 12：基于语法的数据读入.....	17
3.4 任务 13：应用设计与开发.....	18
3.4.1 商业表决系统.....	18
3.4.2 代表选举系统.....	19
3.4.3 聚餐点菜系统.....	20
3.5 任务 14：应对面临的新变化.....	21
3.5.1 商业表决应用：可以一次表决多个商业提案.....	21
3.5.2 代表选举应用：遴选规则变化.....	22
3.5.3 聚餐点菜应用：取消权重设置、只计算“喜欢”的票数.....	22
3.6 Git 仓库结构.....	23
4 实验进度记录.....	23
5 实验过程中遇到的困难与解决途径.....	24

6 实验过程中收获的经验、教训、感想.....	24
6.1 实验过程中收获的经验和教训（必答）	24
6.2 针对以下方面的感受（必答）	25

1 实验目标概述

本次实验覆盖课程第 2、3 章的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

- 子类型、泛型、多态、重写、重载
- 继承、委派、CRP
- 语法驱动的编程、正则表达式
- 设计模式

本次实验给定了多个具体应用，学生不是直接针对每个应用分别编程实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的 ADT 及其实现，充分考虑

这些应用之间的相似性和差异性，使 ADT 有更大程度的复用（可复用性）和更

容易面向各种变化（可维护性）。

2 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

在这里给出你的 GitHub Lab3 仓库的 URL 地址（HIT-Lab3-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab3-120L022109.git>

3 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

3.1 待开发的三个应用场景

简要介绍三个应用。

分析三个应用场景的异同，理解需求：它们在哪些方面有共性、哪些方面有差异。

共性：

都有一定数量的候选对象（数量 ≥ 1 ）

都对拟遴选对象数量有一定要求（这个数量是提前确定的）

投票人数量都是提前确定的

都有属于自己的投票类型

差异

投票人类型不同

投票类型不同

是否实名

对选票的合法性判断规则不同

计票规则不同

遴选规则不同

3.2 ADT 识别与设计

该节是本实验的核心部分。

3.2.1 任务 1：投票类型 **VoteType**

对 **VoteType** 的设计

```

public class VoteType {

    // key为选项名、value为选项名对应的分数
    12 usages
    private Map<String, Integer> options = new HashMap<>();

    // Rep Invariants
    // options的size大于0
    // options中的key和value都不为null
    // Abstract Function
    // key为选项名、value为选项名对应的分数
    // Safety from Rep Exposure
    // 没有可以修改或者获取rep的方法
}

```

方法测试

checkLegality

测试不合法输入

测试合法输入

测试不包含的输入

getScoreByOption 方法实现

测试不合法输入

测试包含的 type

测试不包含的 type

方法实现

CheckRep

更具 RI 检查

检查 options 是否为空

检查 options 中的 key 和 value 是否为空

构造方法

将传入的 map 全部加入到 options 中

```

/*
 * 可以自行设计该方法所采用的参数
 */
└ lsxuan12138
    public VoteType(Map<String, Integer> map) { options.putAll(map); }

    /**

```

checkLegality(String option)

若 option 为 null 抛出异常

若 options 中包含 option， 返回 true

反之返回 false

getScoreByOption(String option)

若 option 为 null 抛出异常
 若 option 不是合法的，抛出异常
 否则返回 options.get(option)

3.2.2 任务 2：投票项 VoteItem<C>

对 VoteItem 的设计

```
↳ lsxuan12138 +1
public class VoteItem<C> {

    // 本投票项所针对的候选对象
    3 usages
    private C candidate;
    // 对候选对象的具体投票选项，例如“支持”、“反对”等
    // 无需保存具体数值，需要时可以从投票活动的VoteType对象中查询得到
    3 usages
    private String value;

    // Rep Invariants
    // candidates和value都不为null
    // Abstract Function
    // candidate本投票项所针对的候选对象
    // value对候选对象的具体投票选项，例如“支持”、“反对”等
    // Safety from Rep Exposure
    // getter方法返回不可变的对象，不会造成内存泄露
}
```

方法测试

getVoteValue

创建一个 voteItem，
 测试其 getVoteValue 是否与设定的相同
 检查相同的情况
 检查相同的情况

getCandidate

创建一个 voteItem，
 测试其 getCandidate 是否与设定的相同
 检查相同的情况
 检查相同的情况

方法实现

checkRep

根据 RI
 检查 candidate 是否为 null
 检查 value 是否为 null

构造方法

因为 rep 都是不可变的类型，直接将传入的属性赋值给 rep 即可
 getVoteValue 和 getCandidate 同理直接返回即可

3.2.3 任务 3：选票 Vote

Vote 设计

```

1 inheritor  lsxuan12138 +1
public class Vote<C> implements IVote<C>{

    // 缺省为“匿名”投票，即不需要管理投票人的信息

    // 一个投票人对所有候选对象的投票项集合
    7 usages
    private Set<VoteItem<C>> voteItems = new HashSet<>();
    // 投票时间
    3 usages
    private Calendar date = Calendar.getInstance();

    // Rep Invariants
    // voteItems中没有null
    // Abstract Function
    // voteItems一个投票人对所有候选对象的投票项集合
    // date投票时间
    // Safety from Rep Exposure
    // 防御性拷贝
}

```

方法测试

getVotetems

 检测相同的输入（与设定相同的输入）

 检查不同的输入

candidateIncluded

 检查包含的输入

 检查不包含的输入

方法实现

checRep

 检查 votetems 中是否包含 null

构造方法

 将传入的 set 中的元素全部加入 votetems

getVotetems

 防御性拷贝，返回 votetems 的副本

candidateIncluded

 若 candidate 为 null 抛出异常

 检查 votetems 中是否包含 candidate 和输入相同的元素

3.2.4 任务 4：投票活动 Poll<C>的测试

测试 addCandidates

 测试一般情况（添加的元素中无重复）

测试重复元素
测试 addVoters
 测试一般情况
 测试重复添加同一个元素
测试 setInfo
 利用 observer 方法测试设定的值，是否与设定的相同
测试 addVote
 测试添加合法的选票
 测试添加不合法的选票
测试 statistics
 测试现有投票活动的统计结果与预期是否相同
测试 selection 和 result
 测试经过遴选的结果，使用 result 方法输出结果，检查其与预期是否相同

3.2.5 任务 5：投票活动 **Poll<C>** 的实现类 **GeneralPollImpl**

对 **GeneralPollImpl** 的设计

RI

```
// Rep Invariants
// name!=null date!=null
// 当candidates不为null则其中无重复元素, candidates>=quantity
// 当voters不为null时, voters中key和value都不为null
// quantity>=1
// voteType不为null
// 投票活动的投票数不能大于投票人数
```

AF

```
// Abstract Function
// // 投票活动的名称
// String name;
// // 投票活动发起的时间
// Calendar date;
// // 候选对象集合
// List<C> candidates;
// // 投票人集合, key为投票人, value为其在本次投票中所占权重
// Map<Voter, Double> voters;
// // 拟选出的候选对象最大数量
// int quantity;
// // 本次投票拟采用的投票类型(合法选项及各自对应的分数)
// VoteType voteType;
// // 所有选票集合
// Map<Vote<C>, Boolean> votes;
// // 合法性检查器
// LegitimacyChecker legitimacyChecker;
// // 计票结果, key为候选对象, value为其得分
// Map<C, Double> statistics;
// // 遴选结果, key为候选对象, value为其排序位次
// Map<C, Double> results;
// Safety from Rep Exposure
//
```

CheckRep

将 checkRep 拆分成三部分

分别根据 RI 检查

```
4 usages  lsxuan *
protected boolean checkRep() {
    // name!=null date!=null
    // quantity>=1, quantity<=candidates
    // voteType不为null
    // 当candidates不为null则其中无重复元素,candidates>=1
    // 当voters不为null时, voters中key和value都不为null
    checkCandidates();
    checkVoters();
    checkInfo();
    return true;
}
4 usages  2 overrides  lsxuan
```

```

    4 usages 2 overrides lsxuan
protected void checkInfo(){
    if(name==null||date==null||quantity<1||voteType==null)assert false;
    if (candidates.size() < quantity) assert false;
}

3 usages 1 override lsxuan +1
protected void checkCandidates() {
    if (candidates.size() < 1) assert false;
    for (int i = 0; i < candidates.size(); i++) {
        for (int j = i + 1; j < candidates.size(); j++) {
            assert !candidates.get(i).equals(candidates.get(j));
        }
    }
}

2 usages lsxuan +1
protected void checkVoters(){
    if(voters.size()!=0){
        for (Voter voter:
             voters.keySet()) {
            if(voter==null)assert false;
            if(voters.get(voter)==null)assert false;
        }
    }
}

```

构造方法

```

*/
4 usages lsxuan12138 +1
public GeneralPollImpl() {
    candidates = new ArrayList<>();
    voters = new HashMap<>();
    votes = new HashMap<>();
    statistics = new HashMap<>();
    results = new HashMap<>();
}

```

public void setInfo(String name, Calendar date, VoteType type, int quantity)

首先检查输入数据合法性，不合法则抛出异常

将输入的属性不可变类型直接赋值，可变类型防御性拷贝

public void addVoters(Map<Voter, Double> voters)

将 voters 中的元素全部加入到 this.votes 中

public void addCandidates(List<C> candidates)

将 candidates 中元素遍历，若 this.candidates 中不包含则加入进去

public void addVote(IVote<C> vote)

检查选票合法性并标记

为此将 rep 中的 votes 改变为 Map<Vote<C>, Boolean>

将检查合法性的功能 delegate 到另外的 ADT 中

```
19 usages 1 override  lsxuan12138 +1
@Override
public void addVote(IVote<C> vote) {
    // 此处应首先检查该选票的合法性并标记，为此需扩展或修改rep
    if(legitimacyChecker==null)throw new RuntimeException("请先设置需要的合法性检查器");
    votes.put(vote,legitimacyChecker.checkLegitimacy(vote,candidates,voteType));
    checkRep();
}
```

public void statistics(StatisticsStrategy ss)

先检查现有选票集的合法性

再统计选票

将这两给功能 delegate 到其他 ADT 中

```
 lsxuan12138 +1
@Override
public void statistics(StatisticsStrategy ss) {
    // 此处应首先检查当前所拥有的选票的合法性
    legitimacyChecker.checkLegitimacy(votes,voters);
    //统计选票
    statistics.putAll(ss.statistics(candidates,votes,voters,voteType));
    checkRep();
}
```

public void selection(SelectionStrategy ss)

将遴选结果放入 result 中

将遴选功能 delegate 到其他 ADT 中

public String result()

将遴选结果排序并输出

```
 lsxuan +1
@Override
public String result() {
    StringBuilder stringBuilder = new StringBuilder();
    List<Map.Entry<C,Double>> list = new ArrayList<>(results.entrySet());

    list.sort(new Comparator<Map.Entry<C, Double>>() {
        //降序排序
        public int compare(Map.Entry<C, Double> o1,
                           Map.Entry<C, Double> o2) {
            return o2.getValue().compareTo(o1.getValue());
        }
    });
    int i=1;
    for (Map.Entry<C,Double> o:
         list) {
        stringBuilder.append(o.getKey().toString()).append("\t").append(i++).append("\n");
    }
    return stringBuilder.toString();
}
```

3.2.6 任务 6：投票活动 Poll<C>的子类型

因为在 GeneralPollImpl 设计时将大部分差异化的功能 delegates 出去，这里只需要对 RI 进行一些处理即可

BusinessVoting

针对这个子类型，规定 quantity 强制为 1，且 candidates.size()=1
重写父类的 setInfo

```
@Override
public void setInfo(String name, Calendar date, VoteType type, int quantity) {
    if(quantity!=1)throw new IllegalArgumentException("quantity!=1");
    super.setInfo(name, date, type, quantity);
}
```

重写父类的 addCandidates

```
3 usages  lsxuan
@Override
public void addCandidates(List<Proposal> candidates) {
    if(candidates.size()!=1){
        throw new IllegalArgumentException();
    }
    super.addCandidates(candidates);
}
4 usages  lsxuan
```

另外其对结果的输出应该是表决是否通过

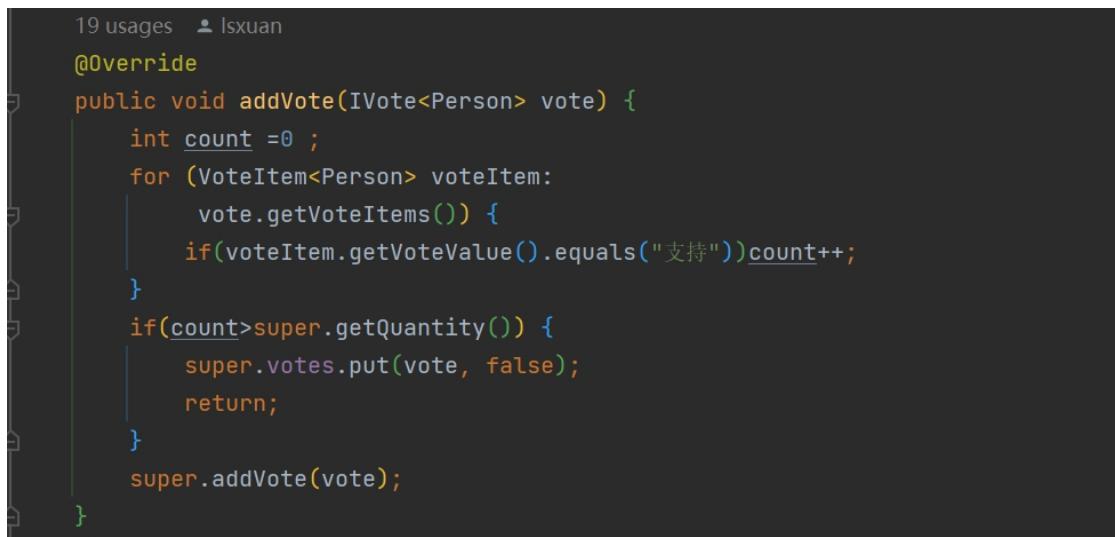
重写 result

```
• lsxuan
@Override
public String result() {
    Map<Proposal, Double> results1 = super.results;
    if(results1.size()!=1)return "表决未通过";
    else return "表决通过, 占比"+results1.values().iterator().next()*100+"%";
```

Election

针对这个子类型，不需要修改其 RI，父类已经全部满足
但是对其选票的合法性有其他要求

重写 addVote



```

19 usages  lsxuan
@Override
public void addVote(IVote<Person> vote) {
    int count = 0;
    for (VoteItem<Person> voteItem:
        vote.getVoteItems()) {
        if(voteItem.getVoteValue().equals("支持"))count++;
    }
    if(count>super.getQuantity()) {
        super.votes.put(vote, false);
        return;
    }
    super.addVote(vote);
}

```

因为这个要求是该应用特有的，不再单独建立 ADT，只在这里处理

DinnerOrder

对这个子类型，比较特殊的是其 quantity 满足
`voters.size()<=quantity<=voters.size()+5<=candidates.size()`
 那么需要对 quantity 和 candidates 做出限制



```

4 usages  lsxuan
@Override
protected void checkInfo() {
    super.checkInfo();
    if(!(super.getVoters().size()<=super.getQuantity()&&super.getQuantity()<=super.getVoters().size()+5))throw new IllegalArgumentException();
}

4 usages  lsxuan
@Override
protected void checkCandidates() {
    super.checkCandidates();
    if(!(super.getVoters().size()+5<=super.getCandidates().size()))assert false;
}

```

3.3 ADT 行为的设计与实现

3.3.1 任务 7：合法性检查

在前面任务中，将合法性检查的功能 delegate 到其他 ADT 中，那么在这个任务中，我们就需要一套对应的 ADT

首先，设定接口，其中有两个方法，分别检查单张选票的合法性和选票集的合法性（如图）

```


    /**
     * 合法性检查
     */
    7 usages 2 implementations  lsxuan12138 +1
    public interface LegitimacyChecker {
        /**
         * 检查一张选票的合法性
         * @param vote 要检查的选票，不应为null
         * @param candidates 所有的候选人，用来判断合法性，不应为null
         * @param voteType 选票类型，用来判断合法性，不应为null
         * @return 返回vote的合法性。合法返回true，反之为false
         * @param <C> 泛型参数
         */
        1 usage 1 implementation  lsxuan
        <C>boolean checkLegitimacy(IVote<C> vote, List<C> candidates, VoteType voteType);

        /**
         * 检查全部选票合法性
         *
         * @param votes 检查的选票集，结果将直接标记在其中
         * @param <C> 泛型参数
         */
        1 usage 2 implementations  lsxuan
        <C> void checkLegitimacy(Map<IVote<C>, Boolean> votes, Map<Voter, Double> voters);
    }
}


```

然后需要构建其对应的子类

对于不记名投票来讲构建 AnonymousGeneralLegitimacyChecker

检查单张选票

检查是否包含不在本次投票活动中的候选人

检查是否出现跳出投票不允许出现的选项值

检查是否出现对同一个候选对象的多次投票

检查一张选票中是否出现本次投票中的所有候选人

检查选票集是否合法

若尚有投票人还未投票，则不能开始计票；

因这是不记名投票，无法确定是否有人未投票或者多投票

只对选票数精选判断

对于记名投票，构建 RegisteredGeneralLegitimacyChecker，继承 AnonymousGeneralLegitimacyChecker

检查单张选票，其父类中方法依然适用，不再重写

检查选票集，父类中方法不再适用，重写方法

若有人还未投票则不能开始计票

若有人提交了多张票，将其均标记为非法

应用一和三都适用与记名投票的合法性检查

应用二适用与不记名投票的合法检查，另外针对应用二的特殊合法性检查已经在 3.2.6 中详述

3.3.2 任务 8：采用 Strategy 设计模式实现灵活的计票规则

设计一套 ADT 来承担计票的规则

首先，设计统一的接口，其中方法用于计票

```
9 usages 4 implementations ▲ lsxuan12138 +1
public interface StatisticsStrategy {
    4 implementations ▲ lsxuan
    <C> Map<C, Double> statistics(List<C> candidates, Map<IVote<C>, Boolean> votes, Map<Voter, Double> voters, VoteType
}
```

针对不记名投票 AnonymousGeneralStatisticsStrategy

直接计算对应的选票的分值和

对于记名投票 RegisteredGeneralStatisticsStrategy

计算分值时考虑投票人的权重

对于应用一 BusinessVoting

实现自己的特殊子类 BusinessVotingStatistics，只对其中“支持”计票但需要考虑记名投票的权重

对于应用二 Election

实现自己的特殊子类 ElectionStatistics 只对其中“支持”计票且不计算权重

对于应用三

适用与记名投票的规则，不再构建自己的类型

3.3.3 任务 9：采用 Strategy 设计模式实现灵活的遴选规则

设计一套 ADT 来承担遴选的规则

首先，设计统一的接口，其中方法用于计票

```
9 usages 4 implementations ▲ lsxuan12138 +1
public interface StatisticsStrategy {
    4 implementations ▲ lsxuan
    <C> Map<C, Double> statistics(List<C> candidates, Map<IVote<C>, Boolean> votes, Map<Voter, Double> voters, VoteType
}
```

通用的遴选规则

对统计结果降序排序，选出前 quantity 个，若分值相同则排名随机

对于应用一 BusinessVoting

实现自己的特殊子类 BusinessVotingSelection，只有分值>2/3 时才将其放入结果

对于应用二 Election

实现自己的特殊子类 ElectionSelection，先将统计结果降序排序，将前 quantity 个放入 result，若最后一个放入的分值与还未放入的下一个分值相同，则删除 result 中与其分值相同的元素

对于应用三

适用与通用的规则，不再构建自己的类型

3.3.4 任务 10：处理匿名和实名投票

使用子类方式实现

```

/*
23 usages  lsxuan
public class RealNameVote<C> extends Vote<C>{
    //投票者
    6 usages
    private Voter voter;

    /**
     * 创建一个选票对象
     * <p>
     * 可以自行设计该方法所采用的参数
     *
     * @param set
     */
    5 usages  lsxuan
    public RealNameVote(Voter voter, Set<VoteItem<C>> set) {
        super(set);
        this.voter = voter;
    }
}


```

使用装饰器模式实现

首先抽象出一个接口

```

public interface IVote <C>{
    /**
     * 查询该选票中包含的所有投票项
     *
     * @return 所有投票项
     */
    2 implementations  lsxuan
    public Set<VoteItem<C>> getVoteItems();

    /**
     * 一个特定候选人是否包含本选票中
     *
     * @param candidate 待查询的候选人
     * @return 若包含该候选人的投票项，则返回true，否则false
     */
    5 usages  2 implementations  lsxuan
    public boolean candidateIncluded(C candidate);
}


```

令 Vote 实现这个接口

再实现一个装饰器类

```

/*
 * @create: 2022-06-22 18:28
 */
17 usages  lsxuan
public class RealNameDecorator<C> implements IVote<C>{
    3 usages
    private IVote<C> vote;
    2 usages
    private Voter voter;
    4 usages  lsxuan
    public RealNameDecorator(Voter voter,IVote<C> vote) {
        this.voter=voter;
        this.vote = vote;
    }

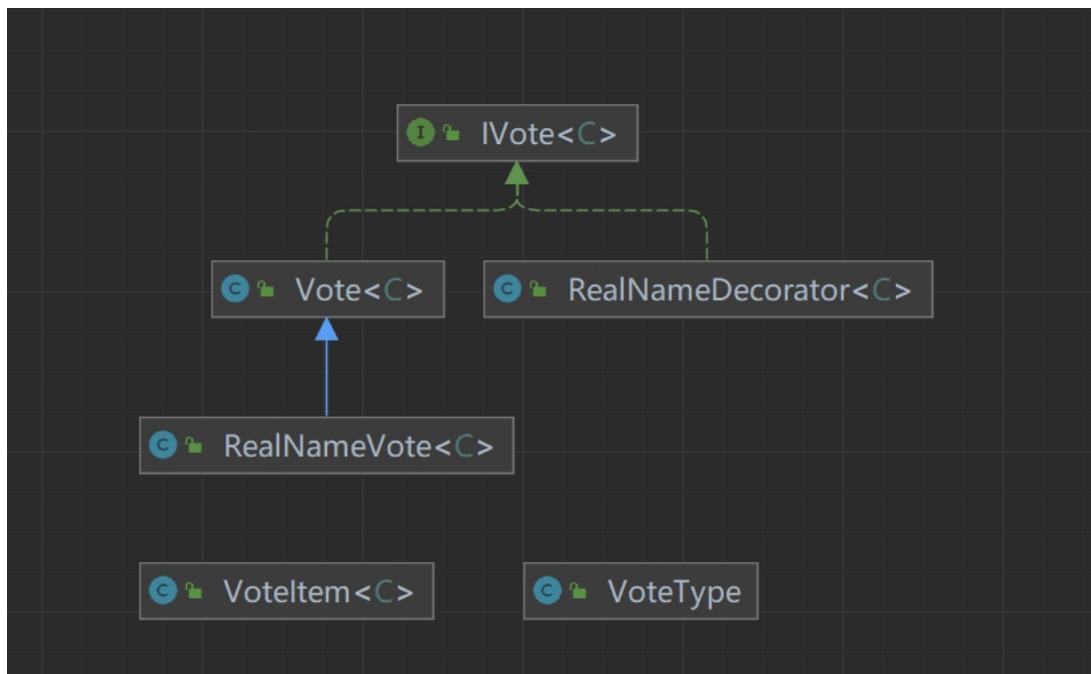
    lsxuan
    @Override
    public Set<VoteItem<C>> getVoteItems() { return vote.getVoteItems(); }

    5 usages  lsxuan
    @Override
    public boolean candidateIncluded(C candidate) { return vote.candidateIncluded(candidate); }

    lsxuan
    public Voter getVoter() {
        return voter;
    }
}

```

最终实现的类结构如图



这里还需要对前面实现的结构和 rep 等进行修改，使用抽象的 IVote 来取代原有的 Vote 例如 GeneralPollImpl 类中

```
// 本次投票拟采用的投票类型（合法选项及各自对应的分数）
5 usages
private VoteType voteType;
// 所有选票集合
protected Map<IVote<C>, Boolean> votes;
4 usages
protected LegitimacyChecker legitimacyChecker;
// 计票结果，key为候选对象，value为其得分
protected Map<C, Double> statistics.
```

还有其他如这样的修改，不一一赘述

对记名投票的演示放在应 3.4 中，这里只简单截图

```
Set<VoteItem<Proposal>> set5 = new HashSet<>();
set5.add(v5);
Vote<Proposal> vote1 = new RealNameVote<>(A, set1);
Vote<Proposal> vote2 = new RealNameVote<>(B, set2);
Vote<Proposal> vote3 = new RealNameVote<>(C, set3);
Vote<Proposal> vote4 = new RealNameVote<>(D, set4);
Vote<Proposal> vote5 = new RealNameVote<>(E, set5);
// 创建投票活动
```

```
vis4.add(vi46);

// 创建6个投票人的选票
IVote<Dish> rv1 = new RealNameDecorator<>(vrGF, new Vote<>(vis1));
IVote<Dish> rv2 = new RealNameDecorator<>(vrF, new Vote<>(vis2));
IVote<Dish> rv3 = new RealNameDecorator<>(vrM, new Vote<>(vis3));
IVote<Dish> rv4 = new RealNameDecorator<>(vrS, new Vote<>(vis4));

// 创建投票活动
```

3.3.5 任务 11：采用 Visitor 设计模式实现功能扩展

先构建一个接口 Visitor

```
import poll.Poll;

5 usages 1 implementation  ↳ lsxuan
public interface Visitor {
    1 usage 1 implementation  ↳ lsxuan
    <C> String visit(Poll<C> poll);
}
```

在 Poll 中增加一个方法

```

    /**
     * 利用Visitor设计模式，为将来的扩展需要留下接口
     *
     * @param visitor 将要扩展功能的实现类的实例对象
     * @return 返回结果
     */
    1 implementation  lsxuan
    public String accept(Visitor visitor);

```

接下来在实现类中实现这个方法

```

    lsxuan
    @Override
    public String accept(Visitor visitor) { return visitor.visit(this); }
}

```

然后实现一个具体的 visitor 类，实现计算合法选票在所有选票中所占比例的功能

```

    * 扩展功能，计算合法选票在所有选票中所占比例
    *
    * @author: lsxuan
    * @email: 1146887979@qq.com
    * @create: 2022-06-23 11:26
    */
    lsxuan
    public class CalculateRatioVisitor implements Visitor{
        1 usage  lsxuan
        @Override
        public <C> String visit(Poll<C> poll) {
            Map<IVote<C>, Boolean> votes = poll.getVotes();
            int count=0;
            for (IVote<C> vote:
                votes.keySet()) {
                if(votes.get(vote))count++;
            }
            double ratio = ((double) count)/ votes.size();
            return "合法选票在所有选票中所占比例为: "+ratio;
        }
    }

```

3.3.6 任务 12：基于语法的数据读入

因为要求支持两种方式的读入，这里准备两种方式的正则表达式

```

public Voter parseString(String regex) {
    Pattern pattern1 = Pattern.compile(regex: "[\\u4e00-\\u9fa5]{1,5}((\\|[\\u4e00-\\u9fa5]{1,5})*)");
    //Pattern pattern2 = Pattern.compile("([\\u4e00-\\u9fa5]{1,5}\\((\\(|(-)?[1-9]\\d*)\\))((\\|\"[\\u4e00-\\u9fa5]{1,5}\\\""
    Pattern pattern2 = Pattern.compile(regex: "[\\u4e00-\\u9fa5]{1,5}\\((\\(|(-)?[1-9]\\d*)\\))((\\|\"[\\u4e00-\\u9fa5]{1,5}\\\"\\\""

```

分别为

“[\\u4e00-\\u9fa5]{1,5}”“[\\u4e00-\\u9fa5]{1,5}”*匹配符合“支持”|“反对”的字符串

`"[\u4e00-\u9fa5]{1,5}"\((0|(-?[1-9]\d*))\)\|\u4e00-\u9fa5]{1,5}"\((0|(-?[1-9]\d*))\)*` 匹配符合“支持”(-1)|“反对”(0)的字符串

当两个都没有匹配到时，抛出异常，若匹配到，则采用对应方式将其解析

```

if(pattern1.matcher(regex).matches()){
    for (String str:
        regex.split( regex: "\\\\|")){
        options.put(str.substring(1,str.length()-1),1);
    }
    return;
}
if (pattern2.matcher(regex).matches()){
    for (String str:
        regex.split( regex: "\\\\|")){
        String[] keyValue = str.split( regex: "\\\\"");
        String key = keyValue[0].substring(1, keyValue[0].length()-1);
        String value = keyValue[1].substring(0,keyValue[1].length()-1);
        options.put(key,Integer.parseInt(value));
    }
    return;
}
throw new IllegalArgumentException();

```

对该构造方法的测试

测试符合表达式“支持”(-1)|“反对”(0)的情况

只有一项的情况

有多项的情况

测试“支持”|“反对”的情况

只有一项的情况

有多项的情况

3.4 任务 13：应用设计与开发

3.4.1 商业表决系统

构造如图的一个商业投票

(1) **商业表决 (BusinessVoting)**: 面向某个商业公司，其内部成员提出某个商业提案（例如“关于投资 xx 项目的提案”），公司董事会的各位董事对其进行实名表决（支持、反对、弃权），各董事在表决中的权重取决于其所持有的公司股票的比例，根据该持股比例对投票结果进行计算，若“支持”票的比例超过 $2/3$ ，则该提案通过，否则该提案不通过。以下给出了针对某提案的表决结果示例：

表 1 商业表决的投票记录示意

董事	股权	投票
A	5%	反对
B	51%	支持
C	10%	支持
D	14%	反对
E	20%	弃权

那么，“支持”所占的比例为 $51\% + 10\% = 61\% < 2/3$ ，故该提案表决未通过。

并设定对应的合法性检查，计票规则，遴选规则

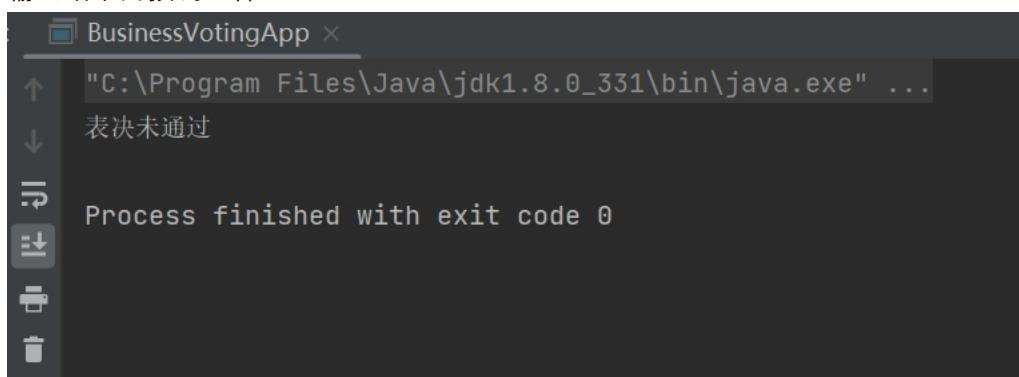
```

poll.addVoters(weighedVoters);
// 设置需要的合法性检查器
poll.setLegitimacyChecker(new RegisteredGeneralLegitimacyChecker());
// 增加投票人的选票
poll.addVote(vote1);
poll.addVote(vote2);
poll.addVote(vote3);
poll.addVote(vote4);
poll.addVote(vote5);

// 按规则计票
poll.statistics(new BusinessVotingStatistics());
// 按规则遴选
poll.selection(new BusinessVotingSelection());
// 输出遴选结果
System.out.println(poll.result());
}

```

输出结果同预测一样



```

BusinessVotingApp
"C:\Program Files\Java\jdk1.8.0_331\bin\java.exe" ...
表决未通过

Process finished with exit code 0

```

3.4.2 代表选举系统

构造一个如图的代表选举

投票人数为6，各自的选票如下：

表 2 代表选举的投票记录示意

候选人	投票人 1	投票人 2	投票人 3	投票人 4	投票人 5	投票人 6
A	支持	反对	弃权	反对	反对	弃权
B	反对	反对	支持	反对	反对	支持
C	支持	支持	支持	反对	支持	支持
D	反对	支持	支持	反对	支持	反对
E	弃权	支持	支持	反对	支持	弃权

并设定对应的合法性检查，计票规则，遴选规则

```

poll.addVoters(weightedVoters);
poll.setLegitimacyChecker(new AnonymousGeneralLegitimacyChecker());
// 增加6个投票人的选票
poll.addVote(rv1);
poll.addVote(rv2);
poll.addVote(rv3);
poll.addVote(rv4);
poll.addVote(rv5);
poll.addVote(rv6);
// 按规则计票
poll.statistics(new ElectionStatistics());
// 按规则遴选
poll.selection(new ElectionSelection());
// 输出遴选结果
System.out.println(poll.result());
}

```

输出结果同预测一样

```

ElectionApp ×
"C:\Program Files\Java\jdk1.8.0_331\bin\java.exe" ...
Person{name='C'} 1
Person{name='E'} 2
Person{name='D'} 3

Process finished with exit code 0

```

3.4.3 聚餐点菜系统

构建一个如图的点菜投票

例如，4个人去吃饭，从菜单的6道菜中选择4道菜，各自偏好如下表所示：

表 3 聚餐点菜的投票记录示意

菜品	爷爷	爸爸	妈妈	儿子	总得分
	4	1	2	2	
A	喜欢	无所谓	喜欢	喜欢	17
B	喜欢	喜欢	不喜欢	无所谓	12
C	无所谓	喜欢	不喜欢	喜欢	10
D	无所谓	喜欢	不喜欢	喜欢	10
E	不喜欢	不喜欢	喜欢	喜欢	8
F	不喜欢	喜欢	不喜欢	不喜欢	2

针对菜品C，有2个“喜欢”票、1个“不喜欢”票，1个“无所谓”票，其总得分为 $4 \times 1 + 1 \times 2 + 2 \times 0 + 2 \times 2 = 10$ 。用相同的计算方式得到其他菜的得分，最终计算排名结果为： $A > B > C = D > E > F$ ，故选择ABCD四道菜。

并设定对应的合法性检查，计票规则，遴选规则

```

poll.addVoters(weightedVoters);
poll.setLegitimacyChecker(new RegisteredGeneralLegitimacyChecker());
// 增加6个投票人的选票
poll.addVote(rv1);
poll.addVote(rv2);
poll.addVote(rv3);
poll.addVote(rv4);
// 按规则计票
poll.statistics(new RegisteredGeneralStatisticsStrategy());
// 按规则遴选
poll.selection(new GeneralSelectionStrategy());
// 输出遴选结果
System.out.println(poll.result());
}

```

输出结果同预期一致

```

DinnerOrderApp x
"C:\Program Files\Java\jdk1.8.0_331\bin\java.exe" ...
Dish{name='A'} 1
Dish{name='B'} 2
Dish{name='C'} 3
Dish{name='D'} 4

```

3.5 任务 14：应对面临的新变化

3.5.1 商业表决应用：可以一次表决多个商业提案

评估之前的设计是否可应对变化、代价如何

可以应对变化，代价很小
 如何修改设计以应对变化
 只需要修改 `BusinessVotingSelection` 中的实现和 `BusinessVoting` 的 `result()` 方法即可

`BusinessVotingSelection` 中直接返回统计结果，不再选择

`Result` 方法

```
lsxuan
@Override
public String result() {
    StringBuilder stringBuilder = new StringBuilder();
    Map<Proposal, Double> results1 = super.results;
    for (Proposal p:
        results1.keySet()) {
        if(results1.get(p)*3>=2)
            stringBuilder.append("表决").append(p.toString()).append(" 通过, 占比").append(results1.get(p) * 100).append("%\n");
        else stringBuilder.append("表决").append(p.toString()).append("未通过, 占比").append(results1.get(p) * 100).append("%\n");
    }
    return stringBuilder.toString();
}
```

3.5.2 代表选举应用：遴选规则变化

评估之前的设计是否可应对变化、代价如何
 可以应对变化，代价较高
 如何修改设计以应对变化
 只需要修改 `ElectionSelection` 即可

```
}
    double d=list.get(i).getValue();
    if(d==list.get(i-1).getValue()){
        for (Map.Entry<C,Double> entry:
            list) {
            if(entry.getValue().equals(d)){
                for (IVote<Person> iVote:
                    poll.getVotes().keySet()) {
                    for (VoteItem<Person> voteItem:
                        iVote.getVoteItems()) {
                        if(voteItem.getVoteValue().equals("反对")){
                            result.put((C) voteItem.getCandidate(), result.get(voteItem.getCandidate())+
                                poll.getType().getScoreByOption(voteItem.getVoteValue()));
                        }
                    }
                }
            }
        }
    }
    //若最后放入的quantity与未放入的下一个值相同，则删除result中所有值相同的元素
}
```

3.5.3 聚餐点菜应用：取消权重设置、只计算“喜欢”的票数

评估之前的设计是否可应对变化、代价如何
 可以应对变化，代价很小

如何修改设计以应对变化

首先将投票人的权重设为一样的值

然后未新建一个计票规则即可

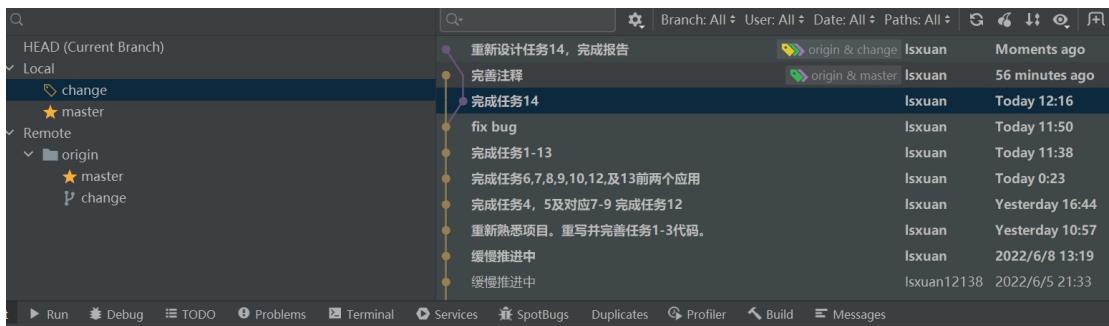
```

protected List<String> validValues = new ArrayList<>();
@Override
public <C> Map<C, Double> statistics(List<C> candidates, Map<IVote<C>, Boolean> votes, Map<VoteItem<C>, Double> result) {
    validValues.add("喜欢");
    Map<C, Double> result = new HashMap<>();
    for (C candidate:
            candidates) {
        result.put(candidate, 0.0);
    }
    for (IVote<C> vote:
            votes.keySet()) {
        for (VoteItem<C> voteItem:
                vote.getVoteItems()) {
            if(validValues.contains(voteItem.getVoteValue())){
                result.put(voteItem.getCandidate(), result.get(voteItem.getCandidate())+
                        voteType.getScoreByOption(voteItem.getVoteValue()));
            }
        }
    }
}

```

3.6 Git 仓库结构

请在完成全部实验要求之后，利用 `Git log` 指令或 `Git` 图形化客户端或 `GitHub` 上项目仓库的 `Insight` 页面，给出你的仓库到目前为止的 `Object Graph`，尤其是区分清楚 `change` 分支和 `master` 分支所指向的位置。



4 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

日期	时间段	计划任务	实际完成情况
2022-06-04	13:30-14:45	任务 1-3	完成
2022-06-04	16:50-18:45	任务 4,5 及 3.4 中任务	未完成, 缓慢进行
2022-06-05	18:30-22:00	任务 4,5 及 3.4 中任务	未完成, 缓慢进行
2022-06-22	10:30-11:00	重新熟悉项目, 重写并完善任务 1-3 中代码	完成
2022-06-22	12:00-16:45	完成任务 4, 5 及对应 7-9, 完成 12	完成
2022-06-22	18:20-17:00	完善任务 12	完成
2022-06-22	21:00-24:00	完成任务 6,7,8,9,10 及 13 前两个应用	完成
2022-06-23	10:00-11:40	完成任务 1-13	完成
2022-06-23	11:40-12:15	完成任务 14	完成
2022-06-23	14:00-17:15	完善注释, 完成报告, 修改 bug	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
对于三个子类型的实现重写方法时 checkRep 会造成循环调用，并栈溢出	拆分 GeneralPollImpl 的 checkRep，并重写规划其中方法对 rep 的影响，使用适当的拆分后的方法
对与记名投票的实现，并根据其实现合法性检查，选票统计	重写之前大部分代码，将具体类型 Vote 换成 IVote，并根据两种不同的实现方法，对方法重写编写
对代表选举应用的修改，其新的遴选规则需要取到反对票数据	使用类似与 visitor 模式的方法，将 poll 对象直接传给 ElectionSelection

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训（必答）

熟悉 Java 中子类型、泛型、多态、重写、重载等语法规则

基本掌握继承、委派、CRP 等常用的软件构造技术

自学正则表达式并熟悉其基本使用

熟悉几种设计模式，并将其应用与代码编写中

6.2 针对以下方面的感受（必答）

- (1) 重新思考 Lab2 中的问题：面向 ADT 的编程和直接面向应用场景编程，你体会到二者有何差异？本实验设计的 ADT 在三个不同的应用场景下使用，你是否体会到复用的好处？

面向 ADT 的编程更加利于代码层面，甚至组件层面的复用，在面对高度相似的使用场景下，面向 ADT 编程能够节省大量的代码量和时间。

- (2) 重新思考 Lab2 中的问题：为 ADT 撰写复杂的 specification, invariants, RI, AF，时刻注意 ADT 是否有 rep exposure，这些工作的意义是什么？你是否愿意在以后的编程中坚持这么做？

编写 specification, invariants, RI, AF，时刻注意 ADT 是否有 rep exposure 等等可以大大减少出现错误的可能，且在方便自己当前开发的同时，也方便了自己和他人的复用

- (3) 之前你将别人提供的 ADT/API 用于自己的程序开发中，本次实验你尝试着开发给别人使用的 ADT/API，是否能够体会到其中的难处和乐趣？

在开发一套 ADT 的过程中，我们需要考虑各种各样的情况，因为我们并不知道 ADT 的用户将会怎样使用我们的 ADT，要将其永远想象为恶意的，我们必须要确保 ADT 在 spec 规定的情况下可以正常的工作。

- (4) 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一个简单的解析器，使用语法和正则表达式去解析一个遵循特定规则的字符串并据此构造对象。你对语法驱动编程有何感受？

语法驱动的编程是需要一定的代价去学习的，且在使用过程中也会有很多的坑。但是他是好用个的，如果对其熟练掌握，就可以以简单的语句实现复杂的功能。

- (5) Lab1 和 Lab2 的工作都不是从 0 开始，而是基于他人给出的设计方案和初始代码。本次实验中也提供了一部分基础代码。假如本实验要求你完全从 0 开始进行 ADT 的设计并用 OOP 实现，你觉得自己是否能够完全搞定？你认为“设计 ADT”的难度主要体现在哪些地方？

如果完全由自己实现，首先这是非常有难度的，虽然可能也许能够实现，但是可能实现的结果并不好用，不够美观。“设计 ADT”的主要难度是在对使用场景中各种事务的共性与个性的分析抽象划分。当我们能很好的划分出共性与个性，那么将其实现也不会非常困难。

- (6) “抽象”是计算机科学的核心概念之一，也是 ADT 和 OOP 的精髓所在。本实验的三个应用既不能完全抽象为同一个 ADT，也不是完全个性化，如何利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、委派、设计模式等技术完成最大程度的抽象和复用，你有什么经验教训？

首先，对于一个抽象的类型（抽象类或者接口）其中所依赖的方法等等，最好也使用抽象类型，这可以方便委派和设计模式等等软件构造方法的使用。

然后关于类的继承，我们可以根据共性来抽象父类，再根据个性来创建子类

(7) 关于本实验的工作量、难度、deadline。

本次实验工作量比较大，难度也很大，deadline 比较合理

(8) 课程结束了，你对《软件构造》课程内容和任课教师的评价如何？

软件构造课程是对我过去所学的编程的一次飞跃，他不再仅仅试试面向一个具体的场景一个过程，而是面向 ADT 的等等更加抽象，更加高层次。他给了我许多经验教训和启发，对以后编程时会有很大帮助。

任课教师授课时专业不失幽默，回答问题及时而清晰。