

**2022年春季学期**

**计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 石卓凡 |
| 学号 | 120L021011 |
| 班号 | 2003001 |
| 电子邮件 | [944613709@qq.com](mailto:944613709@qq.com) |
| 手机号码 | 18974330318 |

**目录**

[1 实验目标概述 1](#_Toc102826659)

[2 实验环境配置 1](#_Toc102826660)

[3 实验过程 1](#_Toc102826661)

[3.1 待开发的三个应用场景 1](#_Toc102826662)

[3.2 ADT识别与设计 1](#_Toc102826663)

[3.2.1 任务1：投票类型VoteType 1](#_Toc102826664)

[3.2.2 任务2：投票项VoteItem<C> 1](#_Toc102826665)

[3.2.3 任务3：选票Vote 2](#_Toc102826666)

[3.2.4 任务4：投票活动Poll<C>的测试 2](#_Toc102826667)

[3.2.5 任务5：投票活动Poll<C>的实现类GeneralPollImpl 2](#_Toc102826668)

[3.2.6 任务6：投票活动Poll<C>的子类型 2](#_Toc102826669)

[3.3 ADT行为的设计与实现 2](#_Toc102826670)

[3.3.1 任务7：合法性检查 2](#_Toc102826671)

[3.3.2 任务8：采用Strategy设计模式实现灵活的计票规则 2](#_Toc102826672)

[3.3.3 任务9：采用Strategy设计模式实现灵活的遴选规则 2](#_Toc102826673)

[3.3.4 任务10：处理匿名和实名投票 2](#_Toc102826674)

[3.3.5 任务11：采用Visitor设计模式实现功能扩展 2](#_Toc102826675)

[3.3.6 任务12：基于语法的数据读入 2](#_Toc102826676)

[3.4 任务13：应用设计与开发 2](#_Toc102826677)

[3.4.1 商业表决系统 2](#_Toc102826678)

[3.4.2 代表选举系统 2](#_Toc102826679)

[3.4.3 聚餐点菜系统 2](#_Toc102826680)

[3.5 任务14：应对面临的新变化 2](#_Toc102826681)

[3.5.1 商业表决应用：可以一次表决多个商业提案 2](#_Toc102826682)

[3.5.2 代表选举应用：遴选规则变化 3](#_Toc102826683)

[3.5.3 聚餐点菜应用：取消权重设置、只计算“喜欢”的票数 3](#_Toc102826684)

[3.6 Git仓库结构 3](#_Toc102826685)

[4 实验进度记录 3](#_Toc102826686)

[5 实验过程中遇到的困难与解决途径 3](#_Toc102826687)

[6 实验过程中收获的经验、教训、感想 4](#_Toc102826688)

[6.1 实验过程中收获的经验和教训（必答） 4](#_Toc102826689)

[6.2 针对以下方面的感受（必答） 4](#_Toc102826690)

# 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）。

# 实验环境配置

配置环境同Lab1，Lab2

使用的IDEA与git

访问https:classroom.github.com/a/ZAJ8w2eC

按照提示建立自己的 Lab3 仓库并关联至自己的学号

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

https:github.com/ComputerScienceHIT/HIT-Lab3-120L021011

# 实验过程

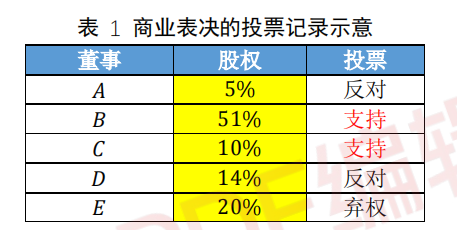
请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## 待开发的三个应用场景

**1.简要介绍三个应用：**

**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。

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

**代表选举（**Election**）：**针对某次活动（例如哈工大学生代表大会），需

要从一群候选人中选出若干人，作为代表参加活动。在该选举中，提前确定一部

分候选人，投票人从已确定的候选人中选取，不可提名新的候选人。计划选出的

代表数量是提前确定的。投票人针对每个候选人匿名选择“支持、反对、弃权”

之一，但选择“支持”的人数不能高于计划选出的代表数量，否则为非法票。

所有投票人的权重均相等。全体投票人投票之后，首先判定各张选票的合法性，

在去除非法选票之后，针对所有候选人，根据其所得到的支持票数量排序，前

个候选人当选；若有多个候选人的支持票数量相等而无法自然排出前名，则仅

有那些明确可进入前名的人当选——这意味着最终选出的人数可能小于。

例如，计划从5个提前确定的候选人（A,B,C,D,E）中选出3个代表，

投票人数为6，各自的选票如下

可以看出，投票人 1、2、4、5、6 的选票都是有效选票，但投票人 3 投了 4

票“支持”，超过了法定当选人数 3，故其选票无效。对有效选票进行统计，候

选人-所得支持票的数目分别为：1、2、5、3、3，候选人、、的票数排名

前三，故成功选出了 3 位代表。如果本次选举希望选出 2 个代表，那么因为和

的得票相同，则只能选出，而和无法当选

**聚餐点菜（**DinnerOrder**）：**一群人去餐馆就餐，需要从该餐馆提供的菜

单中选择若干道菜，点菜的数量要大于等于就餐总人数，且小于总人数+5。每个 人针对菜单上的每一道菜实名表达自己的喜好（喜欢、不喜欢、无所谓），选择 这三个选项的数目无限制。不同人的身份不同，其偏好的影响力会有所不同（例 如家庭聚餐时，老人的权重更大、子女的权重更小，见下表黄色部分）。所有人 表达观点之后，根据影响力加权计票（喜欢、不喜欢、无所谓分别得分 2、0、1）， 取总得分最高的前道菜。若因为有多道菜得分相等而无法自然排出前名，则除 了那些明确可进入前名的菜之外，在其他得分相等的菜中随机选取一部分，凑足个菜。

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：

 针对菜品，有 2 个“喜欢”票、1 个“不喜欢”票，1 个“无所谓”票，其

总得分为4 × 1 + 1 × 2 + 2 × 0 + 2 × 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：，故选择四道菜。

如果最终六道菜的总得分分别为17、12、8、8、8、6，那么前两道菜一定可

以排入前 4 名，但第 3~5 道菜得分都是8，无法区分，此时从中随机选出 2 道菜，

形成了最终入选的 4 道菜

1. **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：**

三个应用中的共性概念

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 概念 | 含义 | 商业表决 | 代表选举 | 聚餐点菜 |
| 投票  活动 | 一组投票人针对一组候选对  象的投票、评价或打分 | 股东表决某个商  业提案 | 学生选举大会  代表 | 一家人聚餐  点菜 |
| 候选  对象 | 被选择/被评价/被打分的具  体事物 | 商业提案 | 被选举的学生 | 菜品 |
| 投票  人 | 根据主观意愿对各候选对象  进行投票、评价或打分的人 | 股东 | 学生 | 就餐者 |
| 投票  项 | 一个投票人对一个候选对象  的具体投票信息 | 一个股东对一个  商业提案的表决  票 | 一个学生对一  个被选举人的  选票 | 一个人对某  个菜品的偏  好 |
| 选票 | 一个投票人对所有候选对象  的投票项的聚合 | 一个股东对所有  商业提案的表决 | 一个学生对所  有被选举人的  投票 | 一个人对所  有菜品的偏 |

三个应用体现差异的维度清单

|  |  |
| --- | --- |
| 维度 | 解释说明 |
| 候选对象的数量m | 有多少个候选对象在一次投票活动中被投票? |
| 拟遴选出的候选对  象数量k | 从所有候选对象中选择出多少个符合规则的候选对象作为最终结  果? |
| 投票类型 | 例如“支持、反对、弃权”、“喜欢、不喜欢、无所谓”，并分别映  射到具体的数值。例如在上述应用中，“支持、反对、弃权”被量  化为1、-1、0，“喜欢、不喜欢、无所谓”被量化为2、O、1。 |
| 实名与否 | 投票记录上是否记录投票人的身份信息?可以是实名，也可以是匿  名。 |
| 投票人权重 | 某个投票人的投票结果在最终计票时所占的比重。可以是所有投票  人有相同的权重，也可以是不同投票人有不同的权重。 |
| 检查合法性 | 每张选票是否合法?以下为非法选票:  一张选票中没有包含本次投票活动中的所有候选人  一张选票中包含了不在本次投票活动中的候选人  一张选票中出现了本次投票不允许的投票选项  一张选票中有对同一个候选对象的多次投票项  所有选票整体是否合法?  若尚有投票人还未投票，则不能开始计票  若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内 |
| 计票规则 | 如何将所有投票人的选票综合到一起，计算出每个候选对象的得票  数或得分，对所有候选人进行排序？ |
| 遴选结果 | 根据何种规则从所有候选对象中选择出一个子集作为最终投票结  果？ |

三个应用在各维度上的异同

|  |  |  |  |
| --- | --- | --- | --- |
|  | 商业表决 | 代表选举 | 聚餐点菜 |
| 候选对象类型 | 商业提案〔例如“关于投资xx项目的  提案”) | 有资格参选的学生（候选人） | 菜品 |
| 候选对象数量m |  | ≥1 | ≥1 |
| 拟追选出的候选对  象数量k | 0（表决未通过）或者1(表决通过） | 提前确定，k ≤m | 提前确定，n≤k ≤n+5≤m.n为投票  人数量 |
| 投票人类型 | 股东 | 学生 | 就餐者 |
| 投票人数量n | 提前确定 | 提前确定 | 提前确定 |
| 投票类型 | 支持/反对/弃权（可看作1/一1/O分） | 支持/反对t弃权（可看作1/一1/0分） | 喜欢/不喜欢/无所谓〔可看作2/0/1分) |
| 实名与否 | 实名 | 匿名 | 实名 |
| 一个投票人在投票  时，不同投票选项  的数量限制 | 无限制 | 支持票数量≤k | 无限制 |
| 投票人权重 | 有，取决于股东所占的股权大小 | 无，所有投票人的权重一样 | 有，就餐者的不同身份具有不同的影响力 |
| 检查合法性 | 表5中给出的合法性检查条件 | 除了表5中给出的合法性检查条件，还需  要检查一个投票人投的支持票总数是否小  于等于k | 表5中给出的合法性检查条件 |
| 计票规则 | 考虑投票人权重，计算提案得“支持”  票的比率 | 计算每个候选人得到的“支持”票总数 | 根据各就餐者的权重，加权求和计算得到  每个菜品的总得分 |
| 诏选规则 | 若超过2/3支持票，则提案通过;否  则，提案不通过 | 根据支持票数量排序，前k个候选人当  选;若有多个候选人的支持票数量相等而  无法自然排出前k名，则仅有那些明确可  进入前k名的人当选 | 根据支持票数量排序，取前k个菜品，若  因为有多道菜得分相等而无法自然排出前  k名，则除了那些明确可进入前k名的菜之  外，在其他得分相等的菜中随机选取一部  分，凑足k个菜。 |

## ADT识别与设计

该节是本实验的核心部分。

### 任务1：投票类型VoteType

**ADT介绍：**

投票类型 VoteType，用于描述在一次投票活动中，投票人对候选对象可

能投出的选票类型。例如在商业表决应用中，投票类型包括“支持、反

对、弃权”，分别对应于 1、0、0 分。在构造投票活动时，需要指定其采

用的投票类型。

**VoteType：**

**属性：**

private Map<String, Integer> options = new HashMap<>();

其中key为选项名、value为选项名对应的分数

Rep Invariants

选项名key不能为""，其中不允许出现空格

options.size()选项个数需要>=2

Abstract Function

AF(String,Integer)->选项其中key为选项名、value为选项名对应的分数

Safety from Rep Exposure

所有的数据域都是私有的

用Collections.unmodifiableMap()转化为不可变类型返回给外部

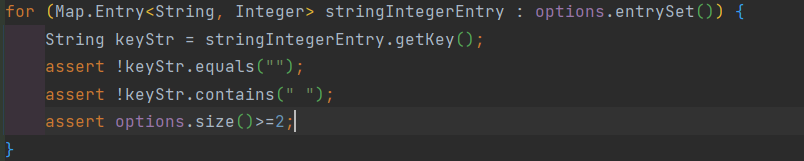
**方法：**

|  |  |
| --- | --- |
| 方法 | 功能 |
| private void checkRep() | 检查不变量 |
| public VoteType(Map<String,Integer> options) | 构造函数 |
| public VoteType(String regex) throws IllegalArgumentException | 根据满足特定语法规则的字符串，创建一个投票类型对象  构造函数 |
| public boolean checkLegality(String option) | 判断一个投票选项是否合法（用于Poll中对选票的合法性检查） |
| public int getScoreByOption(String option) | 根据一个投票选项，得到其对应的分数 |
| public boolean equals(Object o) | 用于判断是否等价 |
| public int hashCode() | 返回哈希值 |
| public Map<String, Integer> getOptions() | 返回options |
| public String toString() | 输出时候转为String模式的自然语言 |

**核心方法实现方案：**

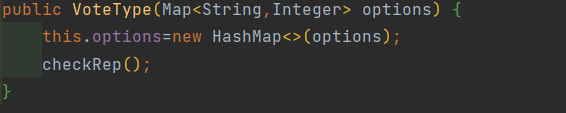
1.private void checkRep()

目的：检查不变量，String不能为""，其中不允许出现空格 options.size()选项个数需要>=2



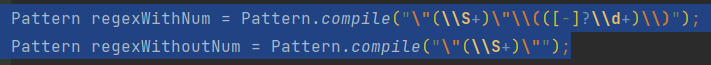
2.public VoteType(Map<String,Integer> options)

构造函数，通过options voteType对应的选项名和分数



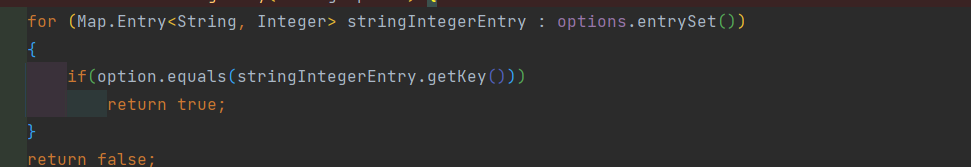
3.public VoteType(String regex) throws IllegalArgumentException

根据满足特定语法规则的字符串，创建一个投票类型对象

1. 通过regex.split根据|来划分options得到inputOptions
2. 给定两个Pattern
3. 
4. 用unMatchFalg记录不匹配次数，如果不匹配次数=0，则最终匹配数字模式，如果不匹配次数=1，则最终匹配无数字模式，如果不匹配此处>=2，则格式不匹配
5. 先尝试匹配有数字模式，利用Matcher.matches(),同时还考虑到如果选项名长度大于5则抛出异常,利用m.group(1)来选定选项名，如果符合要求则将这个加入options
6. 
7. 再次尝试匹配无数字模式利用Matcher.matches(),同时还考虑到如果选项名长度大于5则抛出异常,利用m.group(1)来选定选项名，如果符合要求则将这个加入options
8. 
9. 两次匹配都失败则抛出异常

public boolean checkLegality(String option)

判断一个投票选项是否合法（用于Poll中对选票的合法性检查）

1. 利用forEach循环判断
2. 

public int getScoreByOption(String option)

根据一个投票选项，得到其对应的分数

public Map<String, Integer> getOptions()

Getter返回options

**VoteTypeTest：**

**测试策略：**

针对VoteType(Map<String,Integer> options)

1.不存在投票选项

2.存在投票选项

2.1Key符合条件

2.2Key不符合条件

针对VoteType(String regex)

测试成功：

1.测试带数字的

1.1不含负数

1.2含负数

2.测试不带数字的

测试抛出异常：

1，超过5个长度

2.格式不对

测试checkLegality

1.投票选项为空

2.投票选项不为空

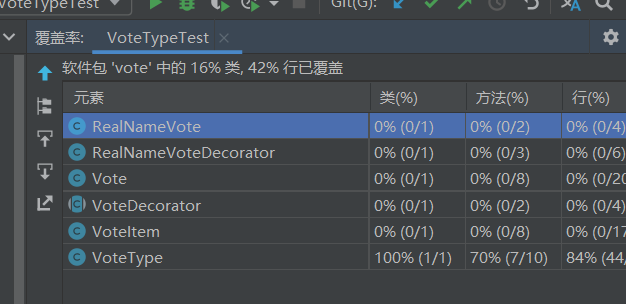
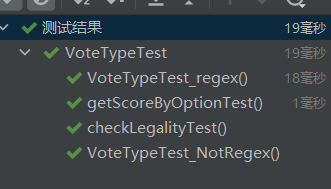
2.1.存在该投票选项

2.2.不存在投票选项

测试getScoreByOption

由前置条件得指必须存在该选项

**测试结果：**



### 任务2：投票项VoteItem<C>

**ADT介绍**

投票记录 VoteItem<C>：代表一个投票人针对一个候选对象的投票结果。

**属性：**

private C candidate;

本投票项所针对的候选对象

private String value;

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

Rep Invariants

选项名value不能为""，其中不允许出现空格.

Abstract Function

AF(candidate,value),对应一次投票，投给候选对象candidate票为value.表征了一

个投票人对一个候选对象的具体评价

Safety from Rep Exposure

所有的数据域都是私有的

**方法：**

|  |  |
| --- | --- |
| 方法 | 功能 |
| private void checkRep() | 检查不变量 |
| public VoteItem | 构造函数，创建一个投票项对象 例如：针对候选对象“张三”，投票选项是“支持” |
| public String getVoteValue() | 得到该投票选项的具体投票项 |
| public C getCandidate() | 得到该投票选项所对应的候选人 |
| public boolean equals(Object o) | 用于判断是否等价 |
| public int hashCode() | 返回哈希值 |
| public String toString() | 输出时候转为String模式的自然语言 |

**核心方法实现方案：**

检查value不为空且不含空格

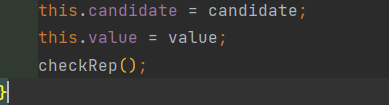
private void checkRep()

关键：assert !value.equals("");

assert !value.contains(" ");

public VoteItem(C candidate, String value)

在初始化之后checkRep（）



**VoteItemTest：**

**测试策略**

voteItem构造函数测试

测试策略对比voteValue与candidate

1.满足条件的value

2.未满足条件的value

2.1value为空

2.2value含空格

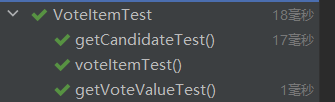
测试getVoteValue

测试策略对比VoteValue

测试getCandidate

对比策略对比Candidate

**测试结果**



### 任务3：选票Vote

**ADT介绍**

选票 Vote，缺省为“匿名”投票，即不需要管理投票人的信息代表一个投票人对所有候选 象的投票结果，由多个投票记录构成。

**属性**

private Set<VoteItem<C>> voteItems = new HashSet<>();

一个投票人对所有候选对象的投票项集合

private Calendar date = Calendar.getInstance();

投票时间

Rep Invariants

voteItems.size()>=1

Abstract Function

AF(voteItem,date),一个投票人对所有候选对象的投票项集合,且记录了投票时间

Safety from Rep Exposure

所有的数据域都是私有的

用Collections.unmodifiableSet()转化为不可变类型返回给外部

使用深拷贝(Calendar) date.clone();

构造函数传入参数时候防御性拷贝

**方法：**

|  |  |
| --- | --- |
| 方法 | 功能 |
| private void checkRep() | 检查不变量 |
| public Vote(Set<VoteItem<C>> voteItems,Calendar date) | 创建一个选票对象 |
| public Vote() | 无参数构造函数  （用于后续的设计模式的构造函数） |
| public Set<VoteItem<C>> getVoteItems() | 查询该选票中包含的所有投票项 |
| public String toString() | 输出时候转为String模式的自然语言 |
| public boolean candidateIncluded(C candidate) | 一个特定候选人是否包含本选票中 |
| public Calendar getDate() | 查询投票日期 |

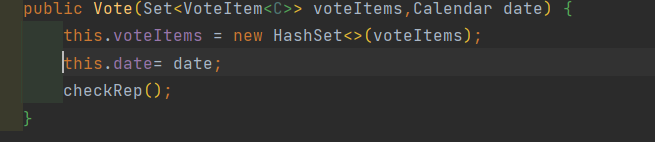
**核心方法实现方案：**

private void checkRep()

检查voteItems.size()>=1

public Vote(Set<VoteItem<C>> voteItems,Calendar date)

在构造完成之后执行checkRep（）



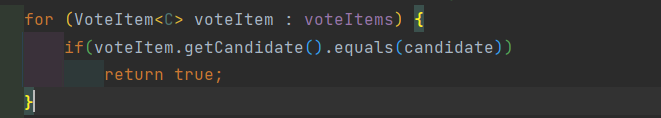
public Set<VoteItem<C>> getVoteItems()

采用Collections.unmodifiableSet防御性拷贝

return Collections.unmodifiableSet(voteItems)

public boolean candidateIncluded(C candidate)

对voteItems进行for循环，在循环内部使用equal对比



**VoteTest**

**测试策略：**

Vote构造函数

1.满足条件

2.不满足条件，voteItem.size()<1

测试getVoteItems

1.voteItems只有一个候选人

2.voteItems含多个候选人

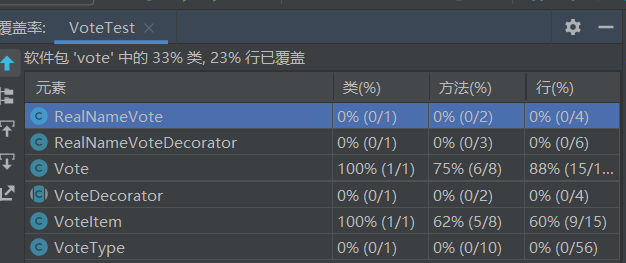
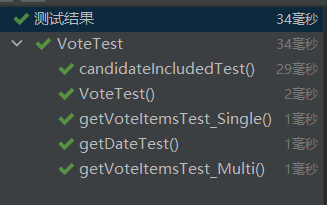
测试candidateIncluded

1.待查询候选人在选票中

2.待查询候选人不在选票中

测试getDate

**测试结果：**



### 任务4：投票活动Poll<C>的测试

**ADT介绍**

投票活动 Poll<C>：代表一次投票活动，C 是表征该投票活动中的“候

选对象”类型的泛型参数。Poll<C>接口中定义的方法是 ADT 所有实例对象都统一具有的“行为”。

**额外加入内容：**

public void accept(Visitor<C> visitor);扩展访问者visitor模式

public void checkVote(Vote<C> vote,Voter voter);在addVote之前检查该选票合法性并标记

public void addVote(Vote<C> vote,Voter voter)修改参数额外加入voter，但是在Poll及GeneralPollImpl中没有记录vote与voter一一对应关系，只记录了voter与投票次数，voter与是否合法，且没有getter方法可以查找vote的voter，因此可以仍然保持为匿名投票（采用Vote类）

public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException;加入了抛出自定义异常CanNotVoteException适用于还有人没有投票导致无法计票

**方法：**

|  |  |
| --- | --- |
| 方法 | 功能 |
| public static <C> Poll<C> create() | 创建投票活动，但尚未设定任何属性，也未有任何投票数据 |
| public void setInfo(String name, Calendar date, VoteType type, int quantity) | 设定投票活动的基本属性 |
| public void addCandidates(List<C> candidates); | 设定候选人 |
| public void addVoters(Map<Voter, Double> voters); | 设定投票人及其权重 |
| public void addVote(Vote<C> vote,Voter voter); | 接收一个投票人对全体候选对象的投票 |
| public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException; | 按规则计票 |
| public void selection(SelectionStrategy<C> ss); | 按规则遴选 |
| public String result(); | 输出遴选结果 |
| public void accept(Visitor<C> visitor); | 扩展访问者visitor模式 |
| public void checkVote(Vote<C> vote,Voter voter); | 在addVote之前检查该选票合法性并标记 |

**PollTest**

**测试策略：**

总体假定条件（如无提示修改就默认）以下

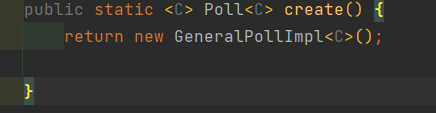
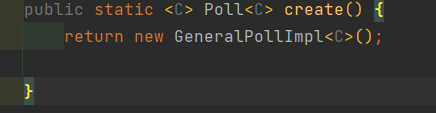
候选人candidate1，candidate2，candidate3

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

测试poll.create

以GeneralPollImpl为实现类



测试setInfo

测试设定之后的name, date, voteType, quantity

addVoter的测试

1.权重都相等

2.权重不都相等

1.候选人vr1，vr2，权重都为1

2.候选人vr1，vr2，权重为1，权重为2

addCandidate的测试

分类：

1.单个候选人，候选人candidate1

2.多个候选人，候选人candidate1，candidate2，candidate3

checkVote测试

四种共用不合法选票的测试

选票不合法情况:

一张选票中没有包含本次投票活动中的所有候选人

一张选票中包含了不在本次投票活动中的候选人

一张选票中出现了本次投票不允许的选项值

一张选票中有对同一个候选对象的多次投票

addVote测试

以2个投票人为例

1.只有单个投票（后续报错无法计票）。

2.多个投票

测试statistics方法

分为正常情况

异常情况1：还有人没有投票无法计票

异常情况2：四种不合法投票（在checkVoteTest以测试）

测试selection

测试策略对比selection得到的result

以Election为实现类代表测试

测试result

测试策略对比result得到的String形式

测试accept方法,访问者Visitor模式

1.全部合法

2.部分合法

/

getVotersVoteFrequencies测试

1.投票次数为1

2.投票次数>1(=2)

3.投票次数<1(=0)

**具体测试样例：**

1.CheckVote测试

选票不合法情况

一张选票中没有包含本次投票活动中的所有候选人

一张选票中包含了不在本次投票活动中的候选人

一张选票中出现了本次投票不允许的选项值

一张选票中有对同一个候选对象的多次投票

候选人candidate1.candidate2

投票人vr1,选票里有candidate1

投票人vr2,选票里有candidate1,candidate3

投票人vr3,选票里有candidate1,candidate2,但是出现了本次投票不允许的 选项值”like"

投票人vr4,选票里有candidate1,candidate1

2. addVote测试

以2个投票人为例

1.只有单个投票（后续报错无法计票）

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support。

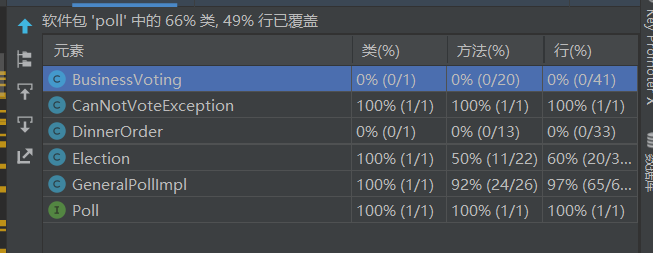
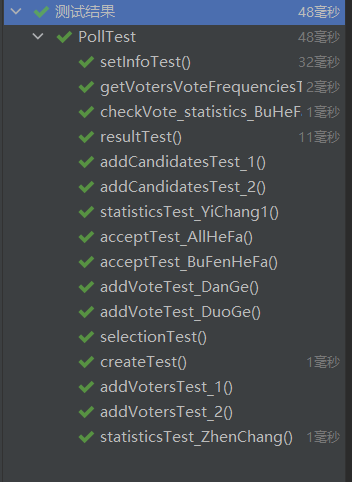
2.多个投票

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support。投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

/

1. 测试accept方法
2. 1.全部合法
3. 选择排名前2的候选人
4. 候选人candidate1，candidate2，candidate3
5. 投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support
6. 投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive
7. 比例2/2
8. 2.部分合法
9. 选择排名前2的候选人
10. 候选人candidate1，candidate2，candidate3
11. 投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support
12. 投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive
13. 投票人vr3，对candidate1-Oppose，对candidate2-Waive
14. 2张合法，3张选票
15. 比例2/3

**测试结果**



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

**ADT介绍**

GeneralPollImpl<C>：是对 Poll<C>的一个实现类。

是包含了三个应用的共性

**关键设计：**

GeneralPollImpl为匿名投票，支持后续子类重写为实名投票

Create以GeneralPollImpl为实现类作为演示

Statistic以ElectionStatisticsStrategy为实现作为演示

Selection以ElectionSelectionStrategy为实现作为演示

**额外加入内容：**

protected Map<Voter,Integer> votersVoteFrequencies;每个voter对应的提交vote次数，key为投票人，value为提交次数，应该每个voter投且仅投1次

protected Map<Vote<C>,Boolean> voteIsLegal;记录每个Vote是否合法或者不合法，key为vote，value为是否合法

public void accept(Visitor<C> visitor);扩展访问者visitor模式

public void checkVote(Vote<C> vote,Voter voter);在addVote之前检查该选票合法性并标记

public void addVote(Vote<C> vote,Voter voter)修改参数额外加入voter，但是在Poll及GeneralPollImpl中没有记录vote与voter一一对应关系，只记录了voter与投票次数，voter与是否合法，且没有getter方法可以查找vote的voter，因此可以仍然保持为匿名投票（采用Vote类）

public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException;加入了抛出自定义异常CanNotVoteException适用于还有人没有投票导致无法计票

**属性：**

投票活动的名称

protected String name;

投票活动发起的时间

protected Calendar date;

候选对象集合

protected 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

protected Map<Voter, Double> voters;

拟选出的候选对象数量

protected 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）

protected VoteType voteType;

所有选票集合

protected Set<Vote<C>> votes;

计票结果，key为候选对象，value为其得分

protected Map<C, Double> statistics;

遴选结果，key为候选对象，value为其排序位次

protected Map<C, Double> results;

每个voter对应的提交vote次数，key为投票人，value为提交次数，应该每个voter投且仅投1次

protected Map<Voter,Integer> votersVoteFrequencies;

记录每个Vote是否合法或者不合法，key为vote，value为是否合法

protected Map<Vote<C>,Boolean> voteIsLegal;

Rep Invariants

name不能为“”

date不能为null

quantity>0;

Abstract Function

AF->为一个匿名投票活动，基本信息有活动名称，活动时间，投票人及权重，拟选出对象数量，投票类型，选票集合，计票结果，遴选结果

每个投票人提交投票次数，投票合法记录。

Safety from Rep Exposure

没有使用public而是protected

对于date可变类型采用深拷贝clone

在addVoter等方法中使用防御性拷贝

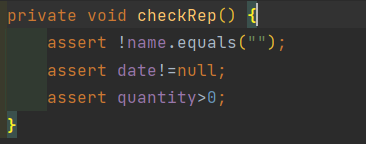
**方法：**

|  |  |
| --- | --- |
| 方法 | 功能 |
| public GeneralPollImpl() | 构造函数 |
| public void setInfo(String name, Calendar date, VoteType type, int quantity) | 设定投票活动的基本属性 |
| public void addCandidates(List<C> candidates); | 设定候选人 |
| public void addVoters(Map<Voter, Double> voters); | 设定投票人及其权重 |
| public void addVote(Vote<C> vote,Voter voter); | 接收一个投票人对全体候选对象的投票，首先检查该选票的合法性并标记并且统计每个voter提交投票次数 |
| public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException; | 按规则计票 |
| public void selection(SelectionStrategy<C> ss); | 按规则遴选 |
| public String result(); | 输出遴选结果 |
| public void accept(Visitor<C> visitor); | 扩展访问者visitor模式 |
| public void checkVote(Vote<C> vote,Voter voter); | 在addVote之前检查该选票合法性并标记 |
| 一系列的getter函数 | 用于获取属性值 |

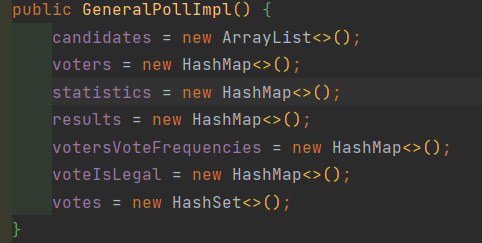
**核心方法实现方案：**

private void checkRep()

检查name不为空，date不为null，quantity>o

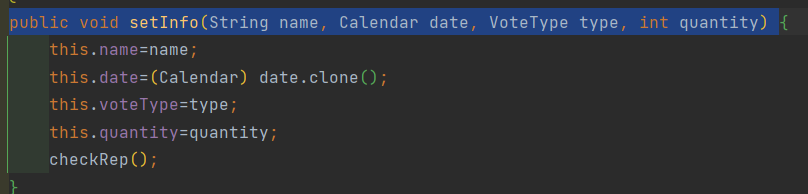


public GeneralPollImpl()



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

对于date使用clone深拷贝，在参数设定之后checkRep（）



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

采用防御性拷贝

public void addCandidates(List<C> candidates)

采用防御性拷贝

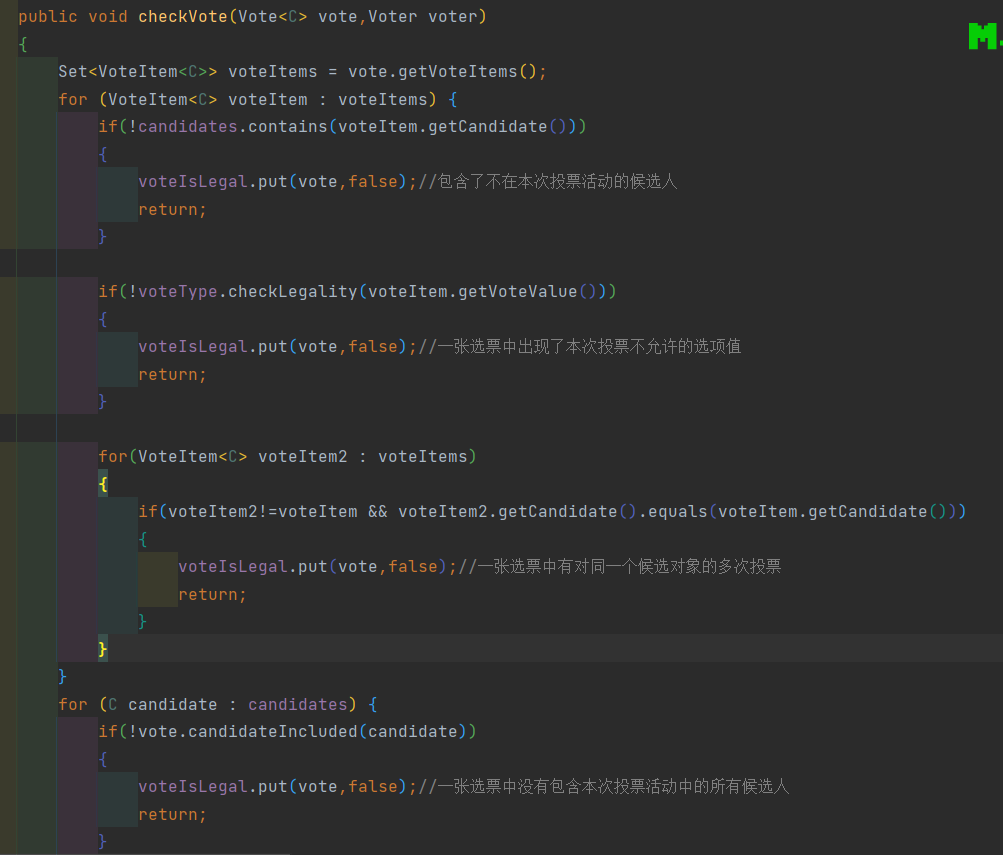
public void checkVote(Vote<C> vote,Voter voter)

在addVote中负责检查该选票合法性并标记

在GeneralPollImpl只负责检查四种通用不合法情况，额外不合法情况在具体应用类中设计

1. 对vote.getVoteItems()进行迭代
2. 对voteItems进行迭代
3. if(!candidates.contains(voteItem.getCandidate()))用来判断包含了不在本次投票活动的候选人
4. if(!voteType.checkLegality(voteItem.getVoteValue()))用来判断一张选票中出现了本次投票不允许的选项值
5. 对循环内部进行二重循环，再利用if(voteItem2!=voteItem && voteItem2.getCandidate().equals(voteItem.getCandidate()))用来判断一张选票中有对同一个候选对象的多次投票
6. 退出上述迭代循环之后
7. 对candidates进行迭代
8. if(!vote.candidateIncluded(candidate))用来判断一张选票中没有包含本次投票活动中的所有候选人
9. 如果都无异常，则为合法

10.检查结束



public void addVote(Vote<C> vote,Voter voter)

接收一个投票人对全体候选对象的投票，首先检查该选票的合法性并标记并且统计每个voter提交投票次数

1. 调用checkVote（）首先检查选票合法性并且标记
2. 无论是否合法（为满足要求任务6若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内）都修改votersVoteFrequncies，记录voter提交投票次数
3. Votes.add(）将本投票记录

public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException

按规则计票

1.检查，若尚有投票人还未投票，则不能开始计票，抛出异常CanNotVoteException

2.在实名投票中检查，若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内。

对于匿名投票，无需检查多次提交的非法规则

3.检查完毕之后执行statistics = ss.statistics（）通过使用 Strategy 模式改造的设计，使 ADT 可 以支持不同的计票规则委托，delegation传入特定的 StatisticsStrateg 子类型对象，即可执行不同的计票方法，分别完成不同的计票规则



public void selection(SelectionStrategy<C> ss)

遴选：

1.直接通过使用 Strategy 模式改造的设计，使 ADT可以支持不同的遴选规则委托，delegation传入特定的 SelectionStrategy 子类型对象，即可执行不同的计票方法，分别完成不同的计票规则

2.关键代码：results = ss.selection(statistics,quantity,voteIsLegal);

public String result()

输出遴选结果

1. 对于得到的results.entrySet()进行迭代
2. 输出格式为：候选对象:"+candidate+" 排名:"+rank.intValue()+ "\n

public void accept(Visitor<C> visitor)

对于Visitor方法的使用

关键代码visitor.visit(this);

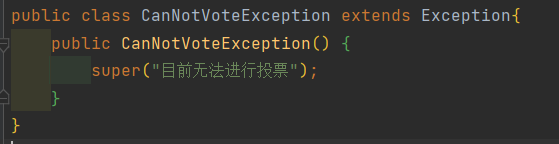
其他方法采用IDEA自动生成

**额外加入类：**

CanNotVoteException

1. public class CanNotVoteException extends Exception

用于计票前查看是否还有人没有投票导致无法计票



**PollTest**

**请见上一节任务5**

### 任务6：投票活动Poll<C>的子类型

**Election：**

匿名投票，代表一次选举活动

继承 GeneralPollImpl<Person> 实现 Poll<Person>

**属性：同父类**

Rep Invariants

未说明部分同父类GeneralPollImpl一致

candidates.size()>=1;

quantity<=candidates.size();

要求VoteType是 支持反对弃权(1|-1|0)

要求每个投票者投的支持票个数<=quantity

Abstract Function

AF(Election)->代表选举，候选对象数量>=1,只允许投票支持反对弃权，匿名投票，没有权重之分，

计票规则统计支持票总数，遴选规则选择排名前k的候选人，若有多个候选人的支持票数量相等而无

法自然排出前k名，则仅有那些明确可进入前k名的人当选；

Safety from Rep Exposure

没有使用public而是protected

对于date可变类型采用深拷贝clone

在addVoter等方法中使用防御性拷贝

**子类关键方法及重写方法：**

private void checkRep()

1.assert candidates.size()>=1;

2.assert quantity<=candidates.size();

public void addCandidates(List<Person> candidates)

1.调用父类方法super.addCandidates(candidates);

2.执行checkRep（）

public void checkVote(Vote<Person> vote, Voter voter)

1. 通过super调用父类的checkVote（）先行检查前四种不合法情况
2. 检查针对应用2的额外不合法情况
   1. 用countSupport计数记录支持票数
   2. 对于vote.getVoteItems进行迭代
   3. 如果匹配到支持票则countSupport++
   4. 对比支持票数量与quantity，若不满足条件则不合法



测试

**测试策略：**

针对Election特殊情况测试

总体正常测试情况1

可以直接选择排名前k的候选人

总体异常测试情况2

当若有多个候选人的支持票数量相等而无法自然排出前k名

Election要求k<=m

测试k（quantity）>m(candidates.size)的报错情况

测试statistics方法

分为正常情况

异常情况1：还有人没有投票无法计票

异常情况2：四种不合法投票（在checkVoteTest以测试）

不合法选票的测试(包括额外的测试)

针对Election的额外不合法情况：支持票数量超过最大数量

测试selection

测试策略对比selection得到的result

1，测试一般情况，可以直接选择排名前k的候选人

2.当若有多个候选人的支持票数量相等而无法自然排出前k名

**具体测试样例**

/

总体正常测试情况1

选择排名前k的候选人

候选人candidate1，candidate2，candidate3

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

/

/

总体异常测试情况2

当若有多个候选人的支持票数量相等而无法自然排出前?名

k=2，应该result只有candidate1

投票人给candidate1 支持票2

candidate2 支持票1

candidate3 支持票1

/

/

Election要求k<=m

测试k（quantity）>m(candidates.size)的报错情况

/

/

测试statistics方法

分为正常情况

异常情况1：还有人没有投票无法计票

异常情况2：四种不合法投票（在checkVoteTest以测试）

不合法选票的测试(包括额外的测试)

选票不合法情况

针对Election的额外不合法情况：支持票数量超过最大数量

<p>

候选人candidate1.candidate2

投票人vr1,选票里有candidate1

投票人vr2,选票里有candidate1,candidate3

投票人vr3,选票里有candidate1,candidate2,但是出现了本次投票不允许的选项值”like"

投票人vr4,选票里有candidate1,candidate1

投票人vr5,选票里有candidate1,candidate2，但是都是支持票，支持票数量超过最大数量

/

/

测试selection

测试策略对比selection得到的result

1，测试一般情况

选择排名前?的候选人

候选人candidate1，candidate2，candidate3

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

/

/

测试selection

当若有多个候选人的支持票数量相等而无法自然排出前?名

k=2，应该result只有candidate1

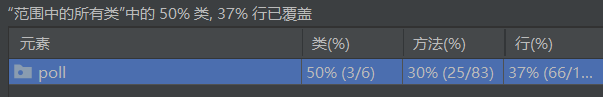
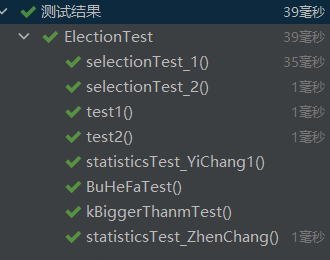
投票人给candidate1 支持票2

candidate2 支持票1

candidate3 支持票1

/

测试结果：



**BusinessVoting：**

**ADT介绍：**

实名投票，代表针对某个商业提案的一次表决

继承 GeneralPollImpl<Proposal> 实现 Poll<Proposal>

**属性：同父类**

/ Rep Invariants

未说明部分同父类GeneralPollImpl一致

candidates.size()=1;

quantity=1;

Abstract Function

AF（businessVoting）->商业提案，候选对象数量为1，拟选出候选对象为0或者1，只允许投票支持否定和弃权，且投票实名的，且能够根据股份划分权重的，

计票规则是统计获得支持票的数量，遴选规则是获得支持票超过合法选票的2/3，即表示表决通过。

Safety from Rep Exposure

没有使用public而是protected

对于date可变类型采用深拷贝clone

在addVoter等方法中使用防御性拷贝

**子类关键方法及重写方法：**

private void checkRep()

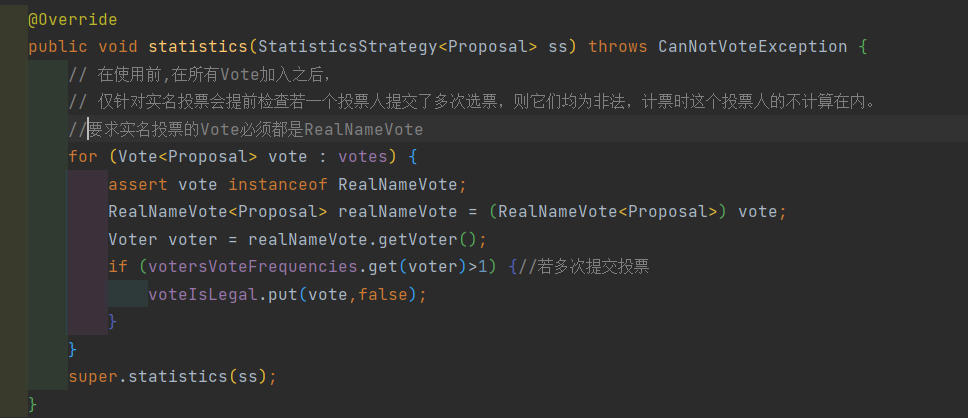
assert super.getQuantity()==1;

assert super.getCandidates().size()==1;

public void statistics(StatisticsStrategy<Proposal> ss) throws CanNotVoteException

针对实名投票会提前检查若一个投票人提交了多次选票，则它们均为非法，计票时这个投票人的不计算在内。

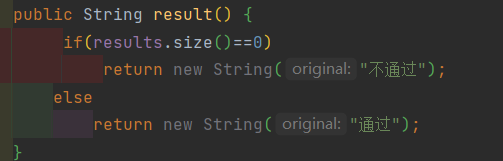
1. 对votes循环
   1. 将vote（已知是实名投票可以向下转型）向下转型为RealNameVote实名投票
   2. 利用RealNameVote.getVoter()得到voter
   3. 利用if (votersVoteFrequencies.get(voter)>1) 判断是否多次提交投票
   4. 如果多次提交投票则不合法



public String result()

将输出的遴选结果根据应用的实际要求

输出结果应该是通过或者不通过



**测试：**

**测试策略：**

/

针对实名投票：

测试对于实名投票测试多次投票后非法的情况

测试statistics

1.权重不全相同

2.权重都相同

selection测试

1.超过2/3,通过

2.刚好等于2/3,未通过

3.小于2/3,未通过

**具体测试样例**

针对实名投票：

对于实名投票测试多次投票后非法的情况

候选人candidate1

投票人vr1,选票里有candidate1

投票人vr1,选票里有candidate1

测试statistics

1.权重不全相同

2.权重都相同

selection测试

1.超过2/3,通过

1个候选对象

候选人candidate1

投票人vr1，权重为3，对candidate1-support

投票人vr2，权重为1，对candidate1-Oppose

投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->3

总共合法选票数量3

均大于(3(2/3))

遴选得到candidate1>1

2.刚好等于2/3,未通过

1个候选对象

候选人candidate1

投票人vr1，权重为2，对candidate1-support

投票人vr2，权重为1，对candidate1-Oppose

投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->2

总共合法选票数量3

2=(3(2/3))

遴选得到 空

3.小于2/3,未通过

1个候选对象

候选人candidate1

投票人vr1，权重为3，对candidate1-Oppose

投票人vr2，权重为1，对candidate1-Oppose

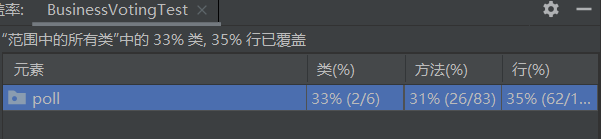
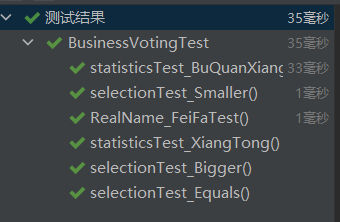
投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->0

总共合法选票数量2

均大于(2(2/3))

遴选得到 空



**DinnerOrder：**

**ADT介绍**

代表一次聚餐点菜活动

继承 GeneralPollImpl<Dish> 实现 Poll<Dish>

**属性：同父类**

Rep Invariants

未说明部分同父类GeneralPollImpl一致

candidates.size()>=1;

voters.size()<=quantity<=voters.size()<=candidates.size()

实名

限定为投票类型 喜欢、不喜欢、无所谓

Abstract Function

AF（businessVoting）->聚餐点菜，候选对象数量>=1，只允许投票喜欢和不喜欢和无所谓，且投票实名的，且能够根据不同身份划分权重的，

计票规则是加权得到各个菜品得分，遴选规则选择排名前k的菜，若因为有多道菜得分相等而无法自然排出前

k名，则除了那些明确可进入前k名的菜之外，在其他得分相等的菜中随

机选取一部分，凑足k个菜

Safety from Rep Exposure

没有使用public而是protected

对于date可变类型采用深拷贝clone

在addVoter等方法中使用防御性拷贝

**子类关键方法及重写方法：**

private void checkRep()

assert candidates.size()>=1;

assert voters.size()<=quantity;

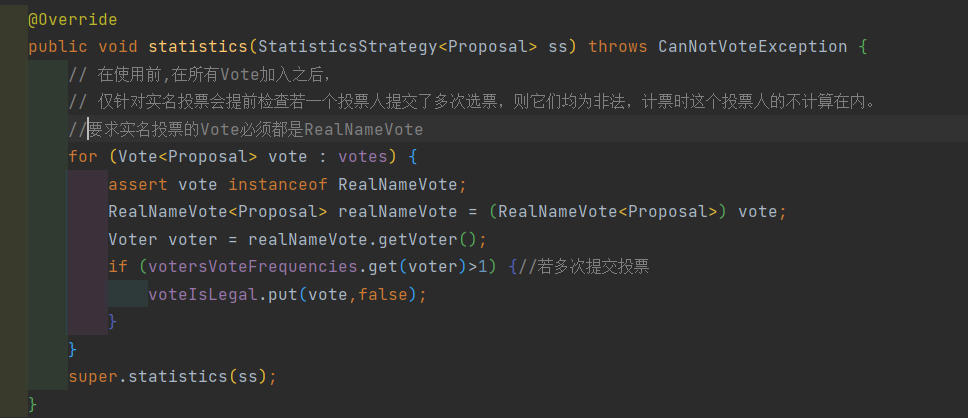
assert quantity<=voters.size();

assert voters.size()<=candidates.size();

public void statistics(StatisticsStrategy<Dish> ss) throws CanNotVoteException

针对实名投票会提前检查若一个投票人提交了多次选票，则它们均为非法，计票时这个投票人的不计算在内。

1. 对votes循环
   1. 将vote（已知是实名投票可以向下转型）向下转型为RealNameVote实名投票
   2. 利用RealNameVote.getVoter()得到voter
   3. 利用if (votersVoteFrequencies.get(voter)>1) 判断是否多次提交投票
   4. 如果多次提交投票则不合法



**Test测试**

**测试策略**

test strategy

针对实名投票：

对于实名投票测试多次投票后非法的情况

statistics测试

1.权重相同

2.权重不相同

selection遴选测试

1.都可以明确进入前k名

2.有部分不可以明确进入

**具体测试样例：**

针对实名投票：

对于实名投票测试多次投票后非法的情况

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票人vr1，对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

投票人vr2，对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

statistics测试

1.权重相同

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，权重为1,对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

2.权重不相同

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

/

/

selection遴选测试

选择排名前k的菜，若因为有多道菜得分相等而无法自然排出前

k名，则除了那些明确可进入前k名的菜之外，在其他得分相等的菜中随

机选取一部分，凑足k个菜

1.都可以明确进入前k名

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

计票得到 candidate1->321=6

计票得到 candidate2->321=6

计票得到 candidate3->111=1

遴选得到candidate2>1

遴选得到candidate1>2

2.有部分不可以明确进入

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-like

投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Unlike

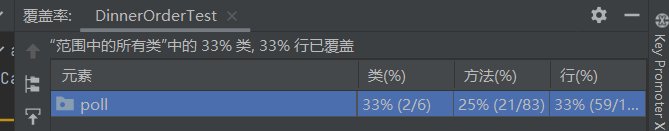
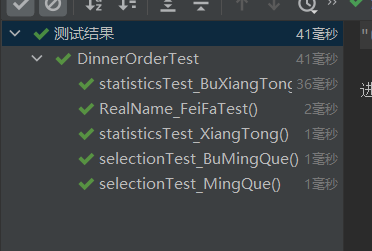
计票得到 candidate1->321=6

计票得到 candidate2->321=6

计票得到 candidate3->6

随机遴选结果！

应该得到candidate1,candidate2,candidate3其中两个



## ADT行为的设计与实现

### 任务7：合法性检查

**通用的四种不合法情况检查：**

一张选票中没有包含本次投票活动中的所有候选人

一张选票中包含了不在本次投票活动中的候选人

一张选票中出现了本次投票不允许的选项值

一张选票中有对同一个候选对象的多次投票

**利用checkVote方法实现**

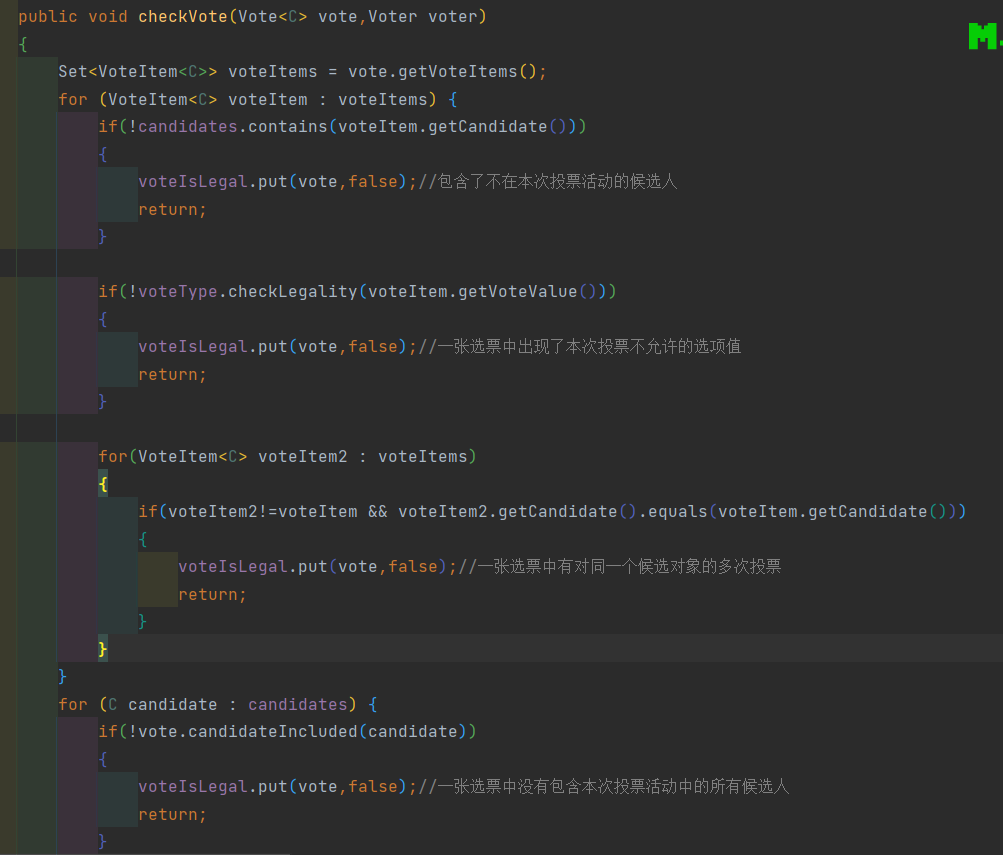
public void checkVote(Vote<C> vote,Voter voter)

在addVote中负责检查该选票合法性并标记

在GeneralPollImpl只负责检查四种通用不合法情况，额外不合法情况在具体应用类中设计

1. 对vote.getVoteItems()进行迭代
2. 对voteItems进行迭代
3. if(!candidates.contains(voteItem.getCandidate()))用来判断包含了不在本次投票活动的候选人
4. if(!voteType.checkLegality(voteItem.getVoteValue()))用来判断一张选票中出现了本次投票不允许的选项值
5. 对循环内部进行二重循环，再利用if(voteItem2!=voteItem && voteItem2.getCandidate().equals(voteItem.getCandidate()))用来判断一张选票中有对同一个候选对象的多次投票
6. 退出上述迭代循环之后
7. 对candidates进行迭代
8. if(!vote.candidateIncluded(candidate))用来判断一张选票中没有包含本次投票活动中的所有候选人
9. 如果都无异常，则为合法

10.检查结束



**Test：**

checkVote测试

四种共用不合法选票的测试

选票不合法情况:

一张选票中没有包含本次投票活动中的所有候选人

一张选票中包含了不在本次投票活动中的候选人

一张选票中出现了本次投票不允许的选项值

一张选票中有对同一个候选对象的多次投票

**具体测试样例**

具体测试样例:

候选人candidate1.candidate2

投票人vr1,选票里有candidate1

投票人vr2,选票里有candidate1,candidate3

投票人vr3,选票里有candidate1,candidate2,但是出现了本次投票不允许的选项值”like"

投票人vr4,选票里有candidate1,candidate1

**测试结果**



**仅应用 2的特殊不合法情况检查：**

一张选票中对所有候选对象的支持票数量大于。

**通过应用2中重写父类方法checkVote进行判断**

public void checkVote(Vote<Person> vote, Voter voter)

1. 通过super调用父类的checkVote（）先行检查前四种不合法情况
2. 检查针对应用2的额外不合法情况
   1. 用countSupport计数记录支持票数
   2. 对于vote.getVoteItems进行迭代
   3. 如果匹配到支持票则countSupport++
   4. 对比支持票数量与quantity，若不满足条件则不合法



**Test测试**

不合法选票的测试(包括额外的测试)

选票不合法情况

针对Election的额外不合法情况：支持票数量超过最大数量

**具体测试样例**

候选人candidate1.candidate2

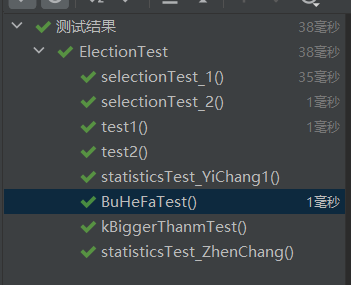
投票人vr1,选票里有candidate1

投票人vr2,选票里有candidate1,candidate3

投票人vr3,选票里有candidate1,candidate2,但是出现了本次投票不允许的选项值”like"

投票人vr4,选票里有candidate1,candidate1

投票人vr5,选票里有candidate1,candidate2，但是都是支持票，支持票数量超过最大数量



### 任务8：采用Strategy设计模式实现灵活的计票规则

在进行计票之前，还需要检查以下内容，具体在 Poll 的 statistics()方法中实现：

**若尚有投票人还未投票，则不能开始计票；**

**在**statistics实现：

1.利用votersVOteFrequencies投票人提交投票次数不等于总投票人数

2.则证明还有投票人尚未投票

3.则抛出异常CanNotVoteException

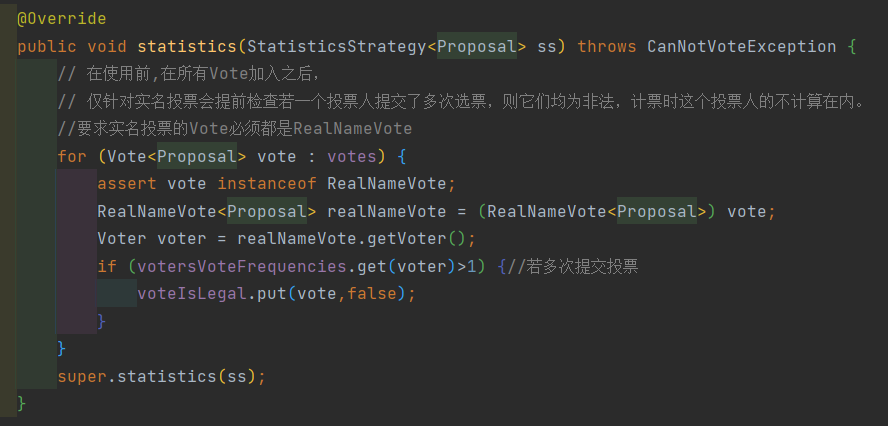
**若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内。**

1. **对于实名投票**（子类BusinessVoting，DinnerOrder）

要在statistics方法中中检查，若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内。

具体重写实现：在statistics方法内部：

1. 对votes循环
   1. 将vote（已知是实名投票可以向下转型）向下转型为RealNameVote实名投票
   2. 利用RealNameVote.getVoter()得到voter
   3. 利用if (votersVoteFrequencies.get(voter)>1) 判断是否多次提交投票
   4. 如果多次提交投票则不合法



**Test测试**

**在BusinessVotingTest测试样例**

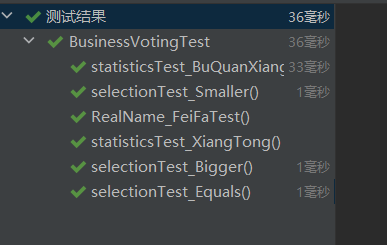
针对实名投票：

对于实名投票测试多次投票后非法的情况

候选人candidate1

投票人vr1,选票里有candidate1

投票人vr1,选票里有candidate1



**在DinnerOrderTest测试样例**

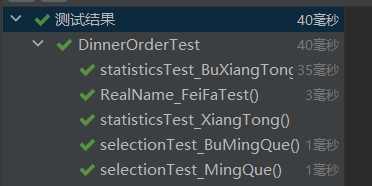
候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票人vr1，对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票人vr1，对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

投票人vr2，对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

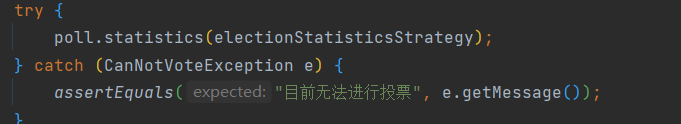


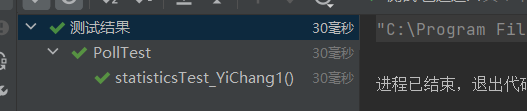
2**. 对于匿名投票**（父类GeneralPollImpl），无需检查多次提交的非法规则



**Test测试**

**测试还有人没投票无法计票**





**综上所述**

public void statistics(StatisticsStrategy<C> ss) throws CanNotVoteException

按规则计票

1.检查，若尚有投票人还未投票，则不能开始计票，抛出异常CanNotVoteException

2.在实名投票中检查，若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内。

对于匿名投票，无需检查多次提交的非法规则

3.检查完毕之后执行statistics = ss.statistics（）通过使用 Strategy 模式改造的设计，使 ADT 可 以支持不同的计票规则委托，delegation传入特定的 StatisticsStrateg 子类型对象，即可执行不同的计票方法，分别完成不同的计票规则

**StatisticsStrategy接口**

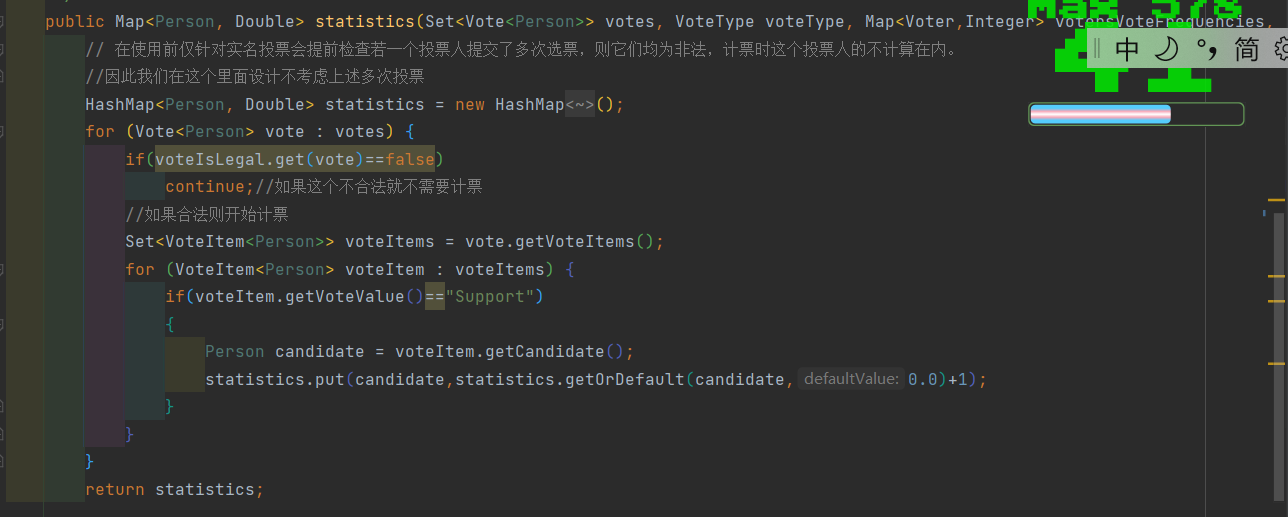
含唯一待实现方法：

Map<C, Double> statistics(.........);

**ElectionStatisticsStrategy <Person> implements StatisticsStrategy<Person>**

规则：无权重计算支持票总总数

1. 对于votes进行循环
   1. 利用voteIsLegal判断这个vote是否合法，如果不合法则不计入计票
   2. 如果合法
      1. 对voteItems进行迭代循环
         1. 如果发现voteItem的投票选项为支持
         2. 则将对应支持人的计数+1



**Test**

**测试statistics方法**

**分为正常情况**

**异常情况1：还有人没有投票无法计票**

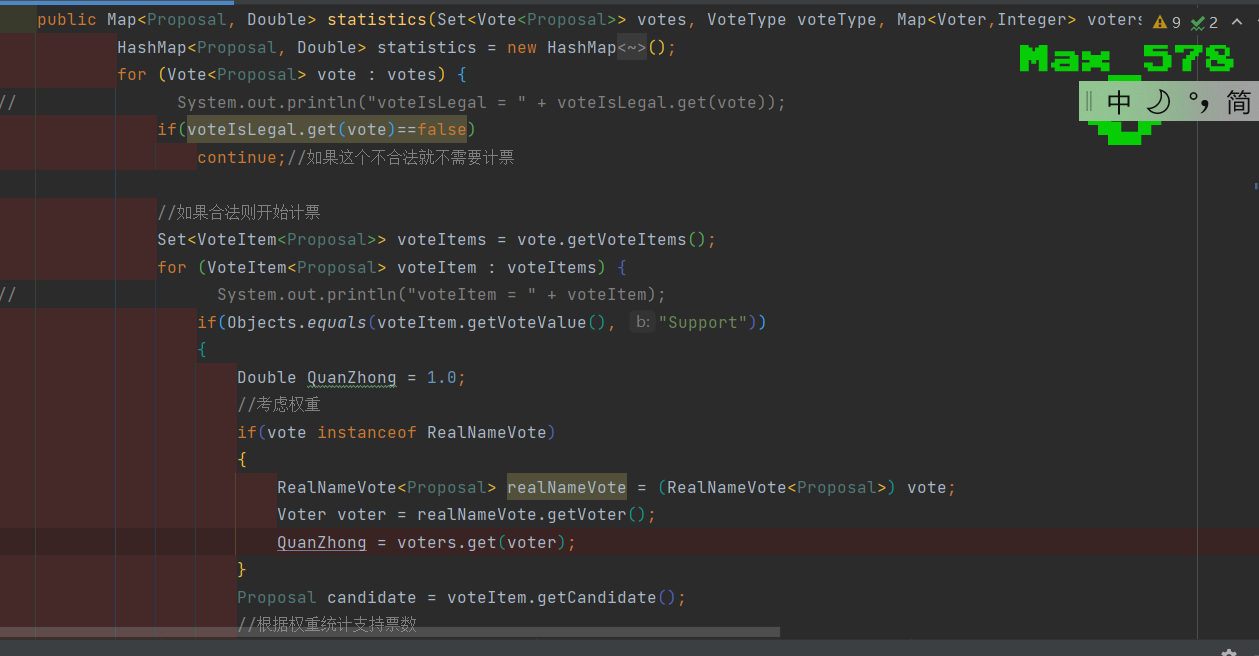
**异常情况2：四种不合法投票（在checkVoteTest以测试）**



**BusinessVotingStatisticsStrategy<Proposal> implements StatisticsStrategy<Proposal>**

规则：统计获得支持票的数量（加权求和）

1. 对于vote进行循环
   1. 利用voteIsLegal进行判断是否合法，如果不合法则不加入计数
   2. 如果合法则对voteItems进行循环
   3. 针对每一个voteItem如果其选项为支持
      1. 先默认权重为1.0（无权重默认为1.0）
      2. 如果vote可以转型为RealNameVote（对于实名投票一定可以转型）
         1. Vote向下转型为RealNameVote
         2. 获得Voter
         3. 通过voters获取权重
      3. 根据权重成相应的比例进行计票支持票

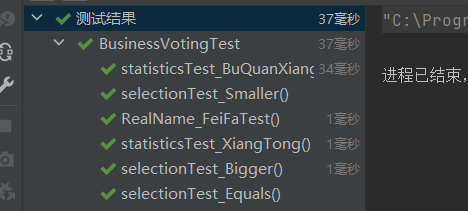


**Test**

**测试statistics**

**1.权重不全相同**

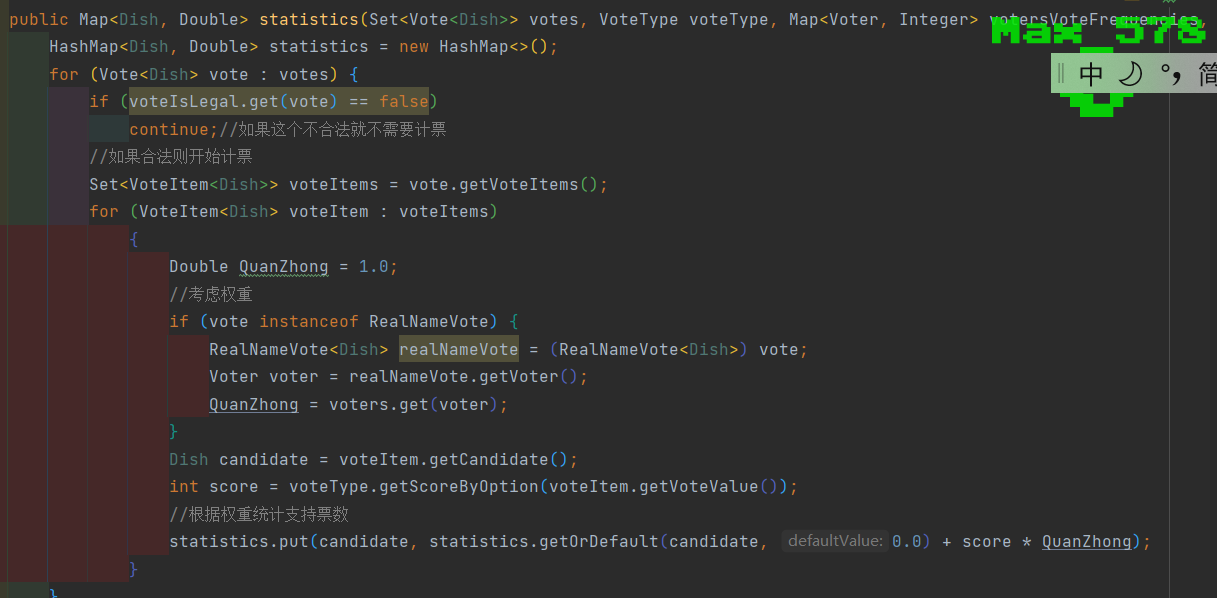
**2.权重都相同**



**DinnerOrderStatisticsStrategy<Dish> implements StatisticsStrategy<Dish>**

规则：统计每个菜品的总得分（加权求和）

1. 对于vote进行循环
   1. 利用voteIsLegal进行判断是否合法，如果不合法则不加入计数
   2. 如果合法则对voteItems进行循环
   3. 针对每一个voteItem如果其选项为支持
      1. 先默认权重为1.0（无权重默认为1.0）
      2. 如果vote可以转型为RealNameVote（对于实名投票一定可以转型）
         1. Vote向下转型为RealNameVote
         2. 获得Voter
         3. 通过voters获取权重
      3. 利用voteType.getScoreByOption()获得该选项对应的分数
      4. 根据权重成相应的比例进行统计加权求和



**Test**

**测试样例：**

**statistics测试**

**1.权重相同**

**选择排名前2的候选对象**

**候选对象candidate1，candidate2，candidate3**

**Like=2,Unlike=0,Indifferent=1**

**投票人vr1，权重为1,对candidate1-Like，对candidate2-Like，对candidate3-Unlike**

**投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent**

**2.权重不相同**

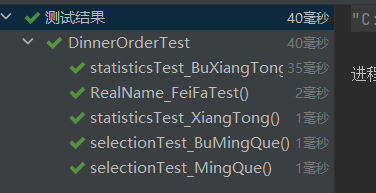
**选择排名前2的候选对象**

**候选对象candidate1，candidate2，candidate3**

**Like=2,Unlike=0,Indifferent=1**

**投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-Unlike**

**投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent**



### 任务9：采用Strategy设计模式实现灵活的遴选规则

**对于public void selection(SelectionStrategy<C> ss)**

关键：results = ss.selection(statistics,quantity,voteIsLegal);

**接口：SelectionStrategy**

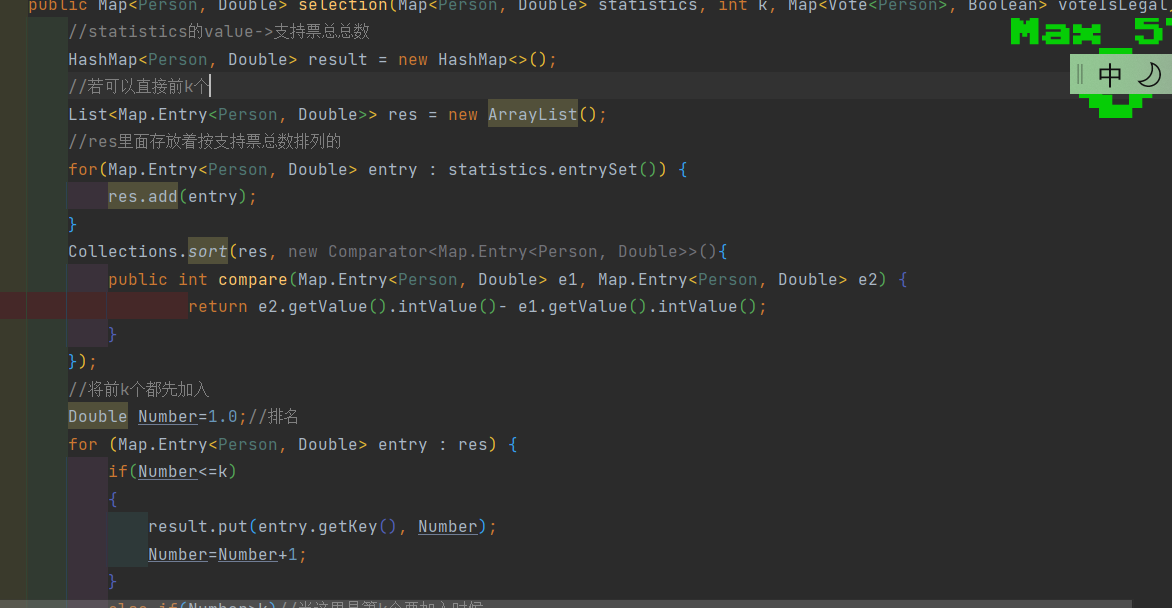
含唯一待实现方法：

Map<C, Double> selection(。。。。);

**ElectionSelectionStrategy<Person> implements SelectionStrategy<Person>**

规则：选择排名前k(quantity)的候选人，若有多个候选人的支持票数量相等而无法自然排出前k名，则仅有那些明确可进入前k名的人当选；

1. 利用List<Map.Entry<Person, Double>> res按照支持票总数排列
   1. 将statistics的计票结果循环将每个元素加入至res
   2. 利用Collections.sort(res, new Comparator<Map.Entry<Person, Double>>利用Comparator重写compare方法通过比较getValue来实现按照支持票总数排列
2. Number记录当前排名
3. 对res内部各个元素循环访问
   1. 将当前排名<=k的时候把相关信息计入result
   2. 一旦超过k了就停止计入
4. 接下来检查是否存在检查是否若有多个候选人的支持票数量相等而无法自然排出前k名，则仅有那些明确可进入前k名的人当选的情况；如果有则删掉对应元素
   1. 如果第k个候选人与第k+1个候选人支持票数量相等，如果是，代表无法自然排出前k名，仅有那些明确可进入前k名的人当选的情况
   2. 把第k个候选人加入待删除名单
   3. 接下来while循环检查前一个候选人是否和第k个候选人支持票数量相等，如果是，代表前面的页不能明确进入前k名，需要加入待删除名单
   4. 将待删除名单里，在result进行删除
5. 留下仅有那些明确可进入前k名的人



**Test**

**测试selection**

**测试策略对比selection得到的result**

**1，测试一般情况可以明确选择排名前k的候选人**

**候选人candidate1，candidate2，candidate3**

**投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support**

**投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive**

**测试selection**

**2.当若有多个候选人的支持票数量相等而无法自然排出前?名**

**k=2，应该result只有candidate1**

**投票人给candidate1 支持票2**

**candidate2 支持票1**

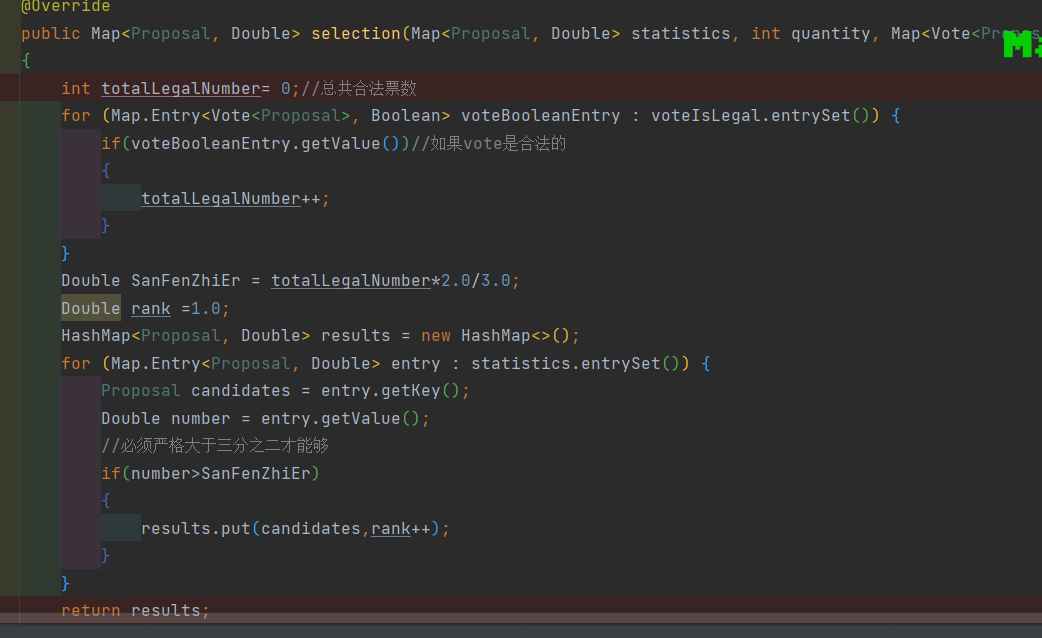
**candidate3 支持票1**



**BusinessSelectionStrategy<Proposal> implements SelectionStrategy<Proposal>**

规则：获得（加权求和之后的）支持票总数，严格（必须大于不能等于）超过（无需考虑加权分配）合法选票个数的2/3，即表示表决通过

1. 初始化总计合法票数=0
2. 对voteIsLeagal.entrySet()进行循环迭代
   1. 如果发现合法则统计合法票数加一
3. 求出合法选票个数的三分之二 SanFenZhiEr
4. 用每一个候选人的获得（加权求和之后的）支持票总数与求出来的合法选票个数的三分之二 SanFenZhiEr进行对比
5. 当严格大于时候加入到results



**Test**

**测试样例**

selection测试

1.超过2/3,通过

1个候选对象

候选人candidate1

投票人vr1，权重为3，对candidate1-support

投票人vr2，权重为1，对candidate1-Oppose

投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->3

总共合法选票数量3

均大于(3(2/3))

遴选得到candidate1>1

2.刚好等于2/3,未通过

1个候选对象

候选人candidate1

投票人vr1，权重为2，对candidate1-support

投票人vr2，权重为1，对candidate1-Oppose

投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->2

总共合法选票数量3

2=(3(2/3))

遴选得到 空

3.小于2/3,未通过

1个候选对象

候选人candidate1

投票人vr1，权重为3，对candidate1-Oppose

投票人vr2，权重为1，对candidate1-Oppose

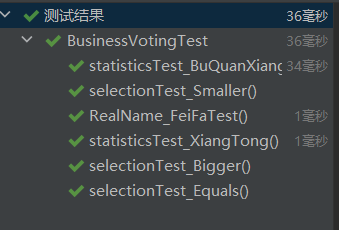
投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->0

总共合法选票数量2

均大于(2(2/3))

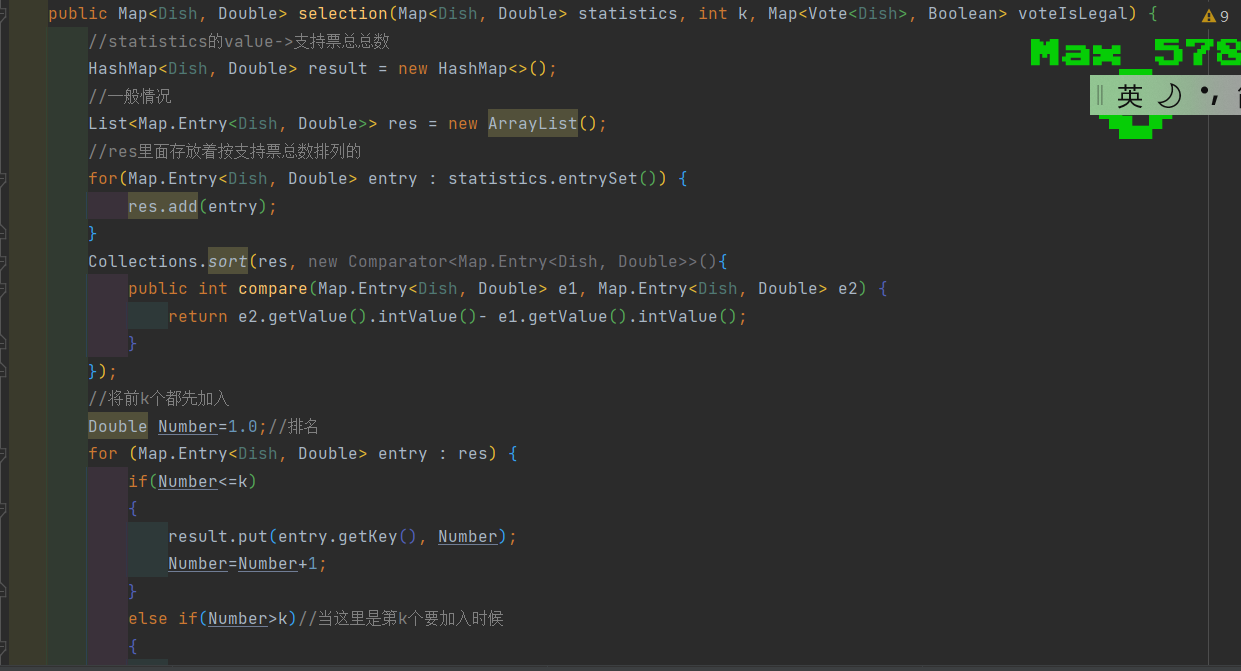
遴选得到 空



**DinnerOrderSelectionStrategy<Dish> implements SelectionStrategy<Dish>**

规则：选择排名前k的菜，若因为有多道菜得分相等而无法自然排出前 k名，则除了那些明确可进入前k名的菜之外，在其他得分相等的菜中随机选取一部分，凑足k个菜

1. 利用List<Map.Entry<Person, Double>> res按照支持票总数排列
   1. 将statistics的计票结果循环将每个元素加入至res
   2. 利用Collections.sort(res, new Comparator<Map.Entry<Person, Double>>利用Comparator重写compare方法通过比较getValue来实现按照支持票总数排列
2. Number记录当前排名
3. 对res内部各个元素循环访问
   1. 将当前排名<=k的时候把相关信息计入result
   2. 一旦超过k了就停止计入
4. 成功加入前k个候选对象（要么明确明可进入前k名的菜，要么在其他得分相等的菜中随机选取一部分，凑足k个菜）



**Test测试样例**

**selection遴选测试**

**选择排名前k的菜，若因为有多道菜得分相等而无法自然排出前**

**k名，则除了那些明确可进入前k名的菜之外，在其他得分相等的菜中随**

**机选取一部分，凑足k个菜**

**1.都可以明确进入前k名**

**选择排名前2的候选对象**

**候选对象candidate1，candidate2，candidate3**

**Like=2,Unlike=0,Indifferent=1**

**投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-Unlike**

**投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent**

**计票得到 candidate1->321=6**

**计票得到 candidate2->321=6**

**计票得到 candidate3->111=1**

**遴选得到candidate2>1**

**遴选得到candidate1>2**

**2.有部分不可以明确进入**

**选择排名前2的候选对象**

**候选对象candidate1，candidate2，candidate3**

**Like=2,Unlike=0,Indifferent=1**

**投票人vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-like**

**投票人vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Unlike**

**计票得到 candidate1->321=6**

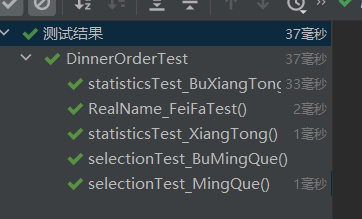
**计票得到 candidate2->321=6**

**计票得到 candidate3->6**

**随机遴选结果！**

**应该得到candidate1,candidate2,candidate3其中两个**

**/**



### 任务10：处理匿名和实名投票

**⚫ 构造 Vote 的子类型 RealNameVote<C>（见 vote 包）**

class RealNameVote<C> extends Vote<C>

属性：同父类

额外多出一个属性：

private Voter voter;投票人

子类特有方法：

相比于父类，将同时记录该投票的投票人信息，实现实名投票

**public RealNameVote(Set<VoteItem<C>> voteItems,Calendar date,Voter voter) {**

**super(voteItems,date);**

**this.voter=voter;**

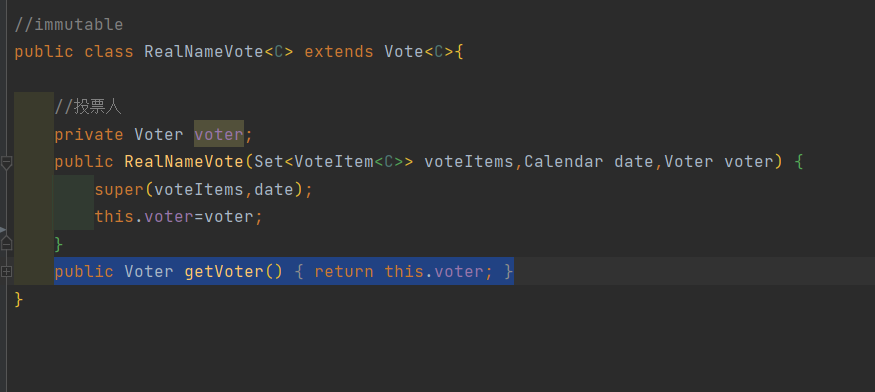
**}**

获得该投票的投票人

**public Voter getVoter() {**

**return this.voter;**

**}**

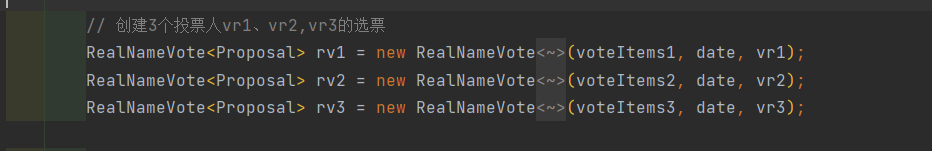


**在应用的客户端程序及Test测试中使用情况：**

**实名选票：**

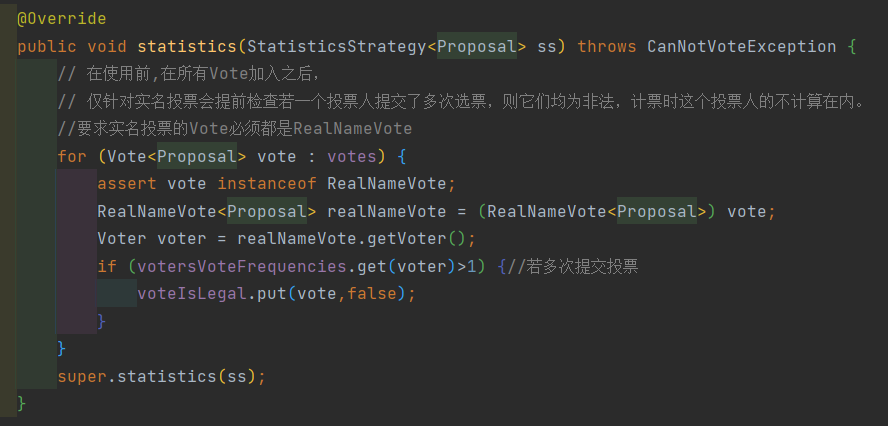
BusinessVotingApp与DinnerOrderApp及BusinessVotingTest，DinnerOrderTest中：

使用的选票是RealNameVote类型实名选票



**实名投票可以实现若一个投票人提交了多次选票，则它们均为非法，计票时不计算在内。**

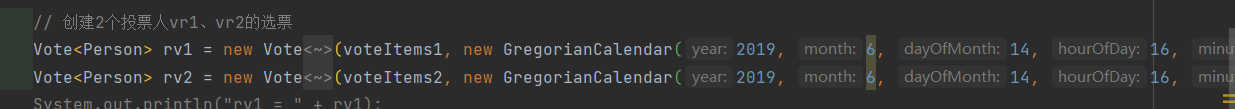
**关键是可以利用**vote（已知是实名投票可以向下转型）向下转型为RealNameVote实名投票，然后RealNameVote.getVoter()得到voter



**匿名选票：**

ElectionApp，ElectionTest，PollTest中

使用的选票是Vote类型匿名选票



**⚫ 使用 Decorator 设计模式，在 Vote 基础上进行扩展。代码中没有给出相**

**应的类，需要自行设计。**

**接口interface VoteInterface<C>**

这个本来应该是要求Poll实现的（根据装饰者模式要求，Decorator抽象类应该要实现Component接口，因此为了切合装饰者模式的设计而新加，需要修改整体设计构造）

即为了代表Decorator设计模式的思路而创建，无实际内容

VoteFuntion代表的是vote应该拥有的方法



**abstract class VoteDecorator<C>implements VoteInterface<C>**

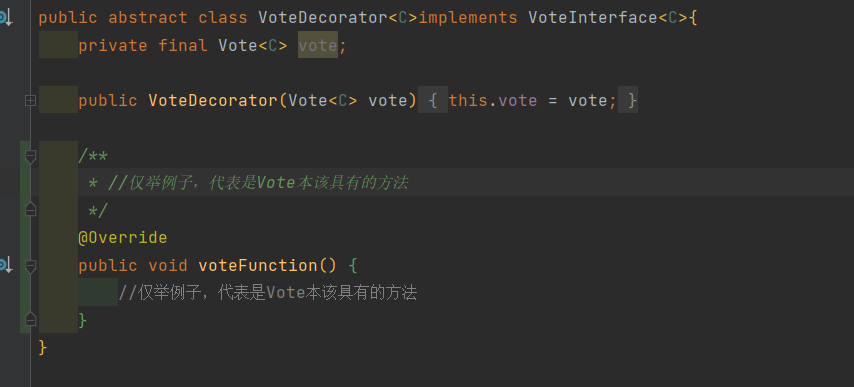
抽象装饰类实现VoteInterface

其中有属性vote

构造方法VoteDecorator(Vote<C> vote) {

this.vote = vote;

}



**class RealNameVoteDecorator<C> extends VoteDecorator<C>**

具体装饰器实现类继承抽象装饰器类

**属性：同父类**

**且额外拥有voter用于实名投票**

**可以传入投票与投票人信息，用于记录实名投票**

**public RealNameVoteDecorator(Vote<C> vote, Voter voter) {**

**super(vote);**

**this.voter = voter;**

**}**

**用于获得投票人信息**

**public Voter getVoter() {**

**return this.voter;**

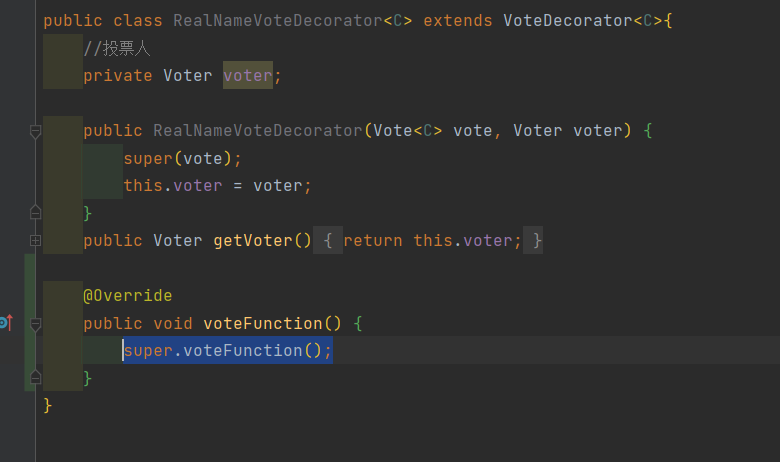
**}**

**代表vote应该具有的方法**

**public void voteFunction() {**

**super.voteFunction();**

**}**



**在应用的客户端程序及Test测试中使用：**

**RealNameVoteDecoratorAPPTest中给出思路**

针对RealNameVoteDecorator仅提供思路由于Vote接口教师框架未给出故只提供思路，无法运行

**客户端关键：**

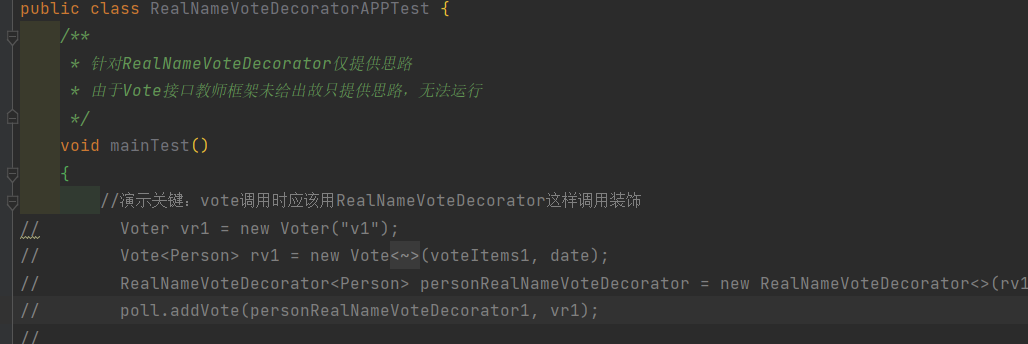
演示关键：vote调用时应该用RealNameVoteDecorator这样调用装饰

Voter vr1 = new Voter("v1");

Vote<Person> rv1 = new Vote<Person>(voteItems1, date);

RealNameVoteDecorator<Person> personRealNameVoteDecorator = new RealNameVoteDecorator<>(rv1, vr1);

poll.addVote(personRealNameVoteDecorator1, vr1);



### 任务11：采用Visitor设计模式实现功能扩展

**abstract class Visitor<C>**

Visitor 设计模式，预留好接口扩展新功能

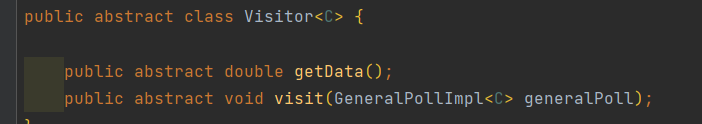
含两个抽象方法：

访问者获取数据

public abstract double getData();

访问者访问

public abstract void visit(GeneralPollImpl<C> generalPoll);



**class CountProportionVisitor<C> extends Visitor<C>**

**属性：**private Double data;合法选票在所有选票中所占比例

方法：

public void visit(GeneralPollImpl<C> generalPoll)



通过访问者计算合法选票在所有选票中所占比例

1. 初始化所有选票与合法选票数=0
2. 循环迭代voteIsLegal.entrySet()
   1. 对于每一张选票，将所有选票数统计++
   2. 对于合法选票，将合法选票数++

3.得到data合法选票在所有选票中所占比例

获得data

public double getData() {

return data;

}

**VisitorTest**

**测试策略：**

测试accept方法,访问者Visitor模式

1.全部合法

2.部分合法

**具体测试：**

1.全部合法

选择排名前2的候选人

候选人candidate1，candidate2，candidate3

投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

比例2/2

2.部分合法

选择排名前2的候选人

候选人candidate1，candidate2，candidate3

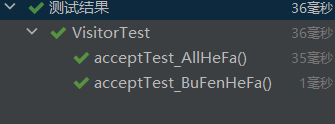
投票人vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票人vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

投票人vr3，对candidate1-Oppose，对candidate2-Waive

2张合法，3张选票

比例2/3

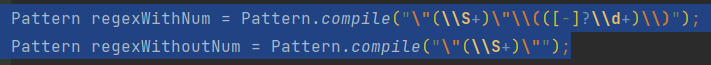


### 任务12：基于语法的数据读入

**实现VoteType函数**

public VoteType(String regex) throws IllegalArgumentException

根据满足特定语法规则的字符串，创建一个投票类型对象

1. 通过regex.split根据|来划分options得到inputOptions
2. 给定两个Pattern
3. 
4. 用unMatchFalg记录不匹配次数，如果不匹配次数=0，则最终匹配数字模式，如果不匹配次数=1，则最终匹配无数字模式，如果不匹配次数>=2，则格式不匹配
5. 先尝试匹配有数字模式，利用Matcher.matches(),同时还考虑到如果选项名长度大于5则抛出异常,利用m.group(1)来选定选项名，如果符合要求则将这个加入options
6. 
7. 再次尝试匹配无数字模式利用Matcher.matches(),同时还考虑到如果选项名长度大于5则抛出异常,利用m.group(1)来选定选项名，如果符合要求则将这个加入options
8. 
9. 两次匹配都失败则抛出异常

**对这个新的构造方法 VoteType(String regex)方法进行完备的测试**

**测试策略：**

针对VoteType(String regex)

测试成功：

1.测试带数字的

1.1不含负数

1.2含负数

2.测试不带数字的

测试抛出异常：

1，超过5个长度

2.格式不对

具体测试：

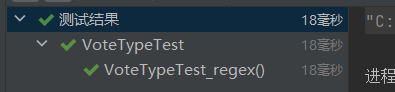
"喜欢"(2)|"不喜欢"(0)|"无所谓"(1)

"喜欢"(-2)|"不喜欢"(0)|"无所谓"(1)

"支持"|"反对"|"弃权"

"超级超级支持"|"反对"|"弃权"

"支持"|(1)"反对"|"弃权"



## 任务13：应用设计与开发

**客户端流程：**

创建投票对象

设定投票对象的权重

设定投票类型

创建候选对象：候选人

创建投票项，投票人vr1对三个候选对象的投票项，后三个是vr2的投票项

创建投票对象vr1、vr2,vr3的选票

创建投票活动

设定投票基本信息：名称、日期、投票类型、选出的数量

增加投票对象及其权重

增加三个投票对象的选票

按规则计票

按规则遴选

输出遴选结果

### 商业表决系统

表决成功->最终这个候选对象被选中

1个候选对象

候选对象candidate1

投票人vr1，权重为3，对candidate1-support

投票人vr2，权重为1，对candidate1-Oppose

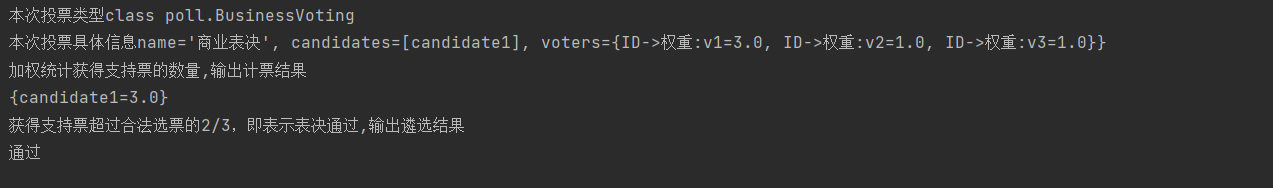
投票人vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->3

总共合法选票数量3

均大于(3(2/3))

遴选得到candidate1>1



### 代表选举系统

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

投票对象vr1，对candidate1-support，对candidate2-oppose，对candidate3-support

投票对象vr2，对candidate1-Oppose，对candidate2-Waive，对candidate3-Waive

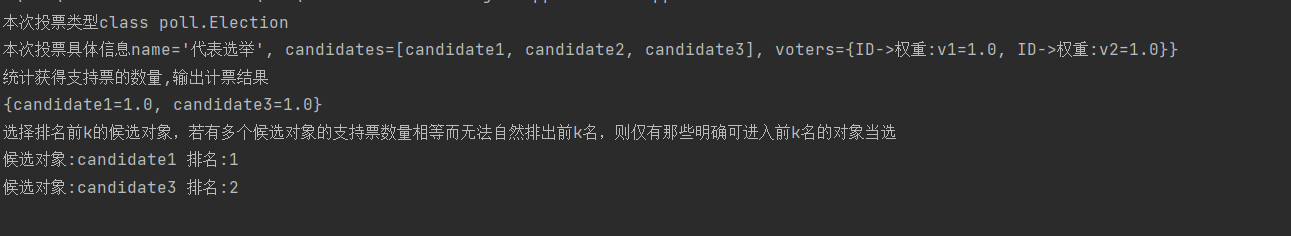
计票结果

candidate1=1.0, candidate3=1.0

遴选结果

候选对象:candidate1 排名:1

候选对象:candidate3 排名:2



### 聚餐点菜系统

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票对象vr1，权重为3,对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票对象vr2，权重为1,对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

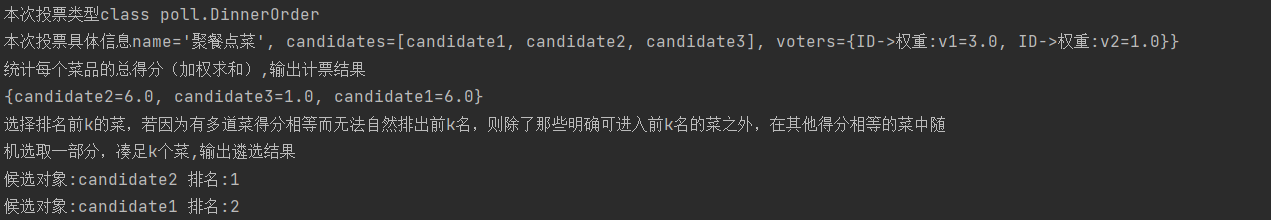
计票得到 candidate1->321=6

计票得到 candidate2->321=6

计票得到 candidate3->111=1

遴选得到candidate2>1

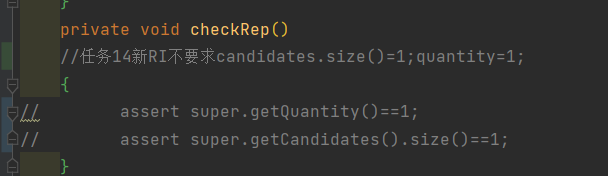
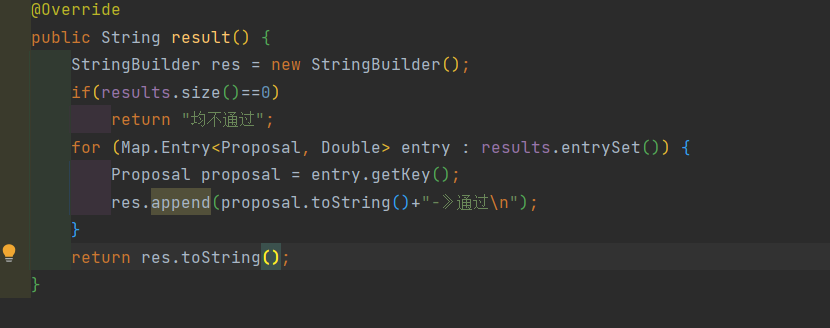
遴选得到candidate1>2



## 任务14：应对面临的新变化

### 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，如何修改设计以应对变化**

1. 之前的设计无法应对变化，改变代价小
2. 改动地方1：
   1. 
   2. checkRep里面删掉对于拟选出候选人最多1个，和候选人只能有1个的RI要求
3. 改动地方2：
   1. 针对result输出遴选结果进行修改，由原来只能判断一个是否通过改为多个候选对象是否通过
   2. 

Task14\_BusinessVotingTest

**测试用例**

1.测试单个候选对象

1个候选对象

候选对象candidate1

投票对象vr1，权重为3，对candidate1-support

投票对象vr2，权重为1，对candidate1-Oppose

投票对象vr3，权重为1，对candidate1-Oppose

计票得到 candidate1->3

总共合法选票数量3

均大于(3(2/3))

遴选得到candidate1>1

2.测试多个候选对象

2个候选对象

候选对象candidate1,candidate2

投票对象vr1，权重为3，对candidate1-support,candidate2->support

投票对象vr2，权重为1，对candidate1-Oppose,candidate2->support

投票对象vr3，权重为1，对candidate1-Oppose,candidate2->support

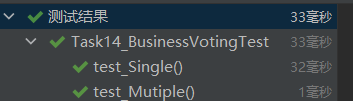
计票得到 candidate1->3,,candidate2->5

总共合法选票数量3

均大于(3(2/3))

遴选得到candidate1,candidate2通过

**APP测试结果**



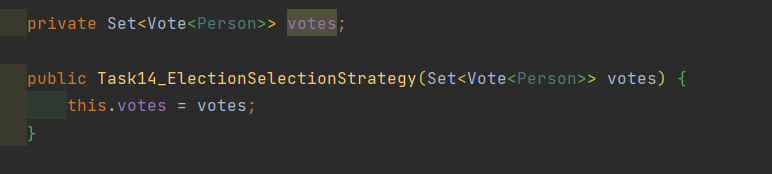
### 代表选举应用：遴选规则变化

**评估之前的设计是否可应对变化、代价如何**

**1.之前的设计无法应对这种变化，需要修改在遴选阶段**

**2.代价特别大！，修改消耗时间2小时，代码修改也大，新增了类**

**如何修改设计以应对变化**

1. **新增类Task14\_ElectionSelectionStrategy**
2. 
3. 额外给定属性votes，构造函数时候传入参数votes
4. 修改selection，具体思路方案

**具体思路：**

**比如k=3,支持票 A 3,B 2,C 1,D 1,E 1**

**反对票 A 0,B 0,C 2,D 1,E 1**

**1.先选出前k个支持票最大的，result为已加入A B D**

**2.查看是否需要比较反对票名单，代比较名单排为D C**

**利用的是第k个之后的元素是否和第k个元素相等，第k个元素之前的是否和第k个元素相等，如果是加入名单**

**3.统计比较名单中的反对票数，根据反对票排序得到D,E,C**

**4.在result中先删掉比较名单上面的所有元素D E C ->result为A B**

**5.重新以此加入比较名单上的元素，直到result元素为k个，加入D -> result为A B D**

**6.检查目前result第k个元素的反对票是否和其在反对票数排序中的下一个元素的反对票数相等，检查到D E反对票相同**

**7.如果相等则删掉上一步重新加入的元素，如果不相等则不用删掉，删掉D ->result为A B**

**8.最终得到结果result为A B**

**Task14\_ElectionTest**

Test测试

情况1:候选人A B C D

投票人1 A弃权，B弃权，C支持，D支持

投票人2 A支持，B支持，C弃权，D弃权

投票人3 A支持，B支持，C弃权，D弃权

投票人4 A支持，B弃权，C反对，D反对

投票人5 A弃权，B弃权，C反对，D弃权

比如k=3,支持票 A 3,B 2,C 1,D 1

反对票 A 0,B 0,C 2,D 1,

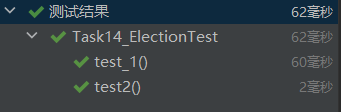
result已加入A B C

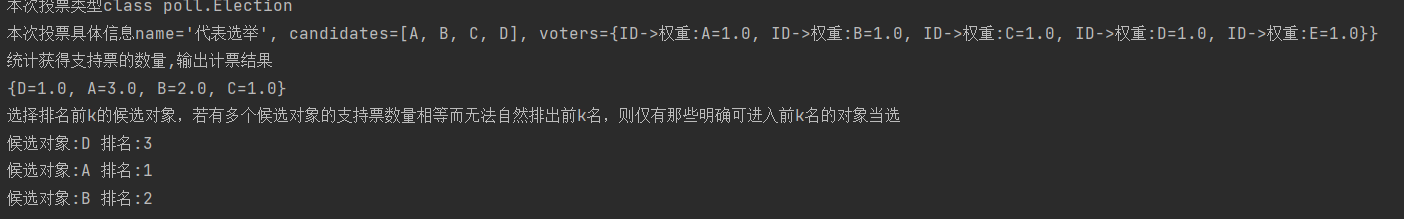
代比较名单排序之后D C

先删掉D C ->result已加入A B

加入D ->result为 A B D

->result为 A B D





情况2

投票人1 A弃权，B弃权，C支持，D支持,E弃权

投票人2 A支持，B支持，C弃权，D弃权,E弃权

投票人3 A支持，B支持，C弃权，D弃权,E弃权

投票人4 A支持，B弃权，C反对，D反对,E弃权

投票人5 A弃权，B弃权，C反对，D弃权,E弃权

投票人6 A弃权，B弃权，C弃权，D弃权,E支持

投票人7 A弃权，B弃权，C弃权，D弃权,E反对

比如k=3,支持票 A 3,B 2,C 1,D 1,E 1

反对票 A 0,B 0,C 2,D 1,E 1

result为已加入A B D

代比较名单排序之后D C

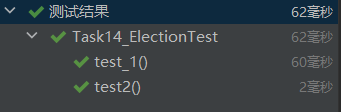
先删掉D E C ->result为已加入A B

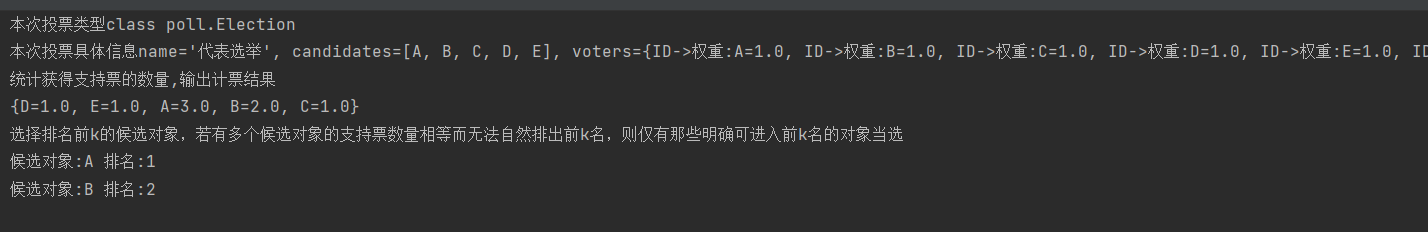
加入D -> result为A B D

检查到D E反对票相同

删掉D ->result为A B

result为A B

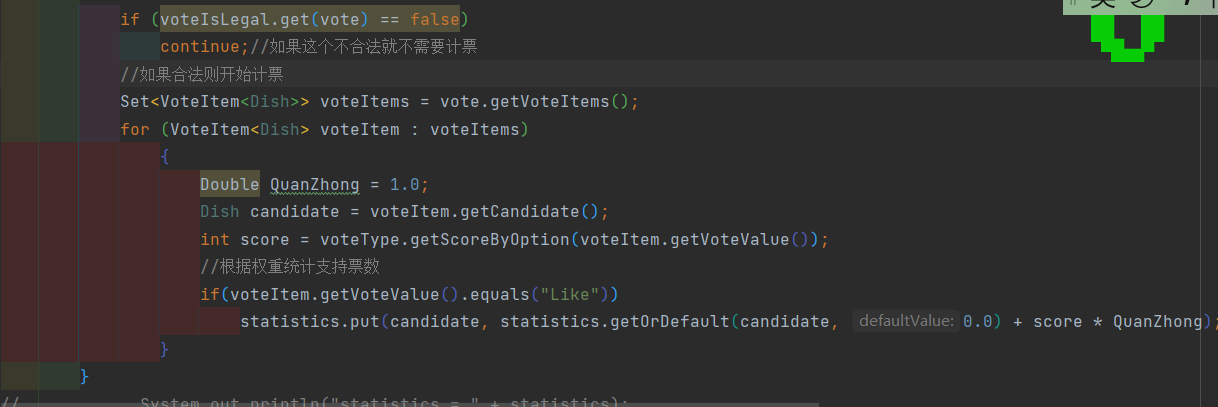




### 聚餐点菜应用：取消权重设置、只计算“喜欢”的票数

**评估之前的设计是否可应对变化、代价如何**

**如何修改设计以应对变化**

1. **之前的设计无法应对变化**
2. **代价较小，利用策略模式新增Task14\_DinnerOrderStatisticsStrategy，且这个类改动也很小**
3. **修改策略：在原有的DinnerOrderStatisticsStrategy基础上，取消权重，将所有权重默认为1，加入对于Like的判断，只有当为Like才加入计数**

测试样例：

选择排名前2的候选对象

候选对象candidate1，candidate2，candidate3

Like=2,Unlike=0,Indifferent=1

投票对象vr1，对candidate1-Like，对candidate2-Like，对candidate3-Unlike

投票对象vr2，对candidate1-Unlike，对candidate2-Unlike，对candidate3-Indifferent

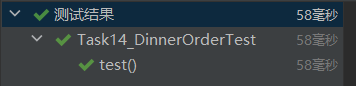
计票得到 candidate1->1

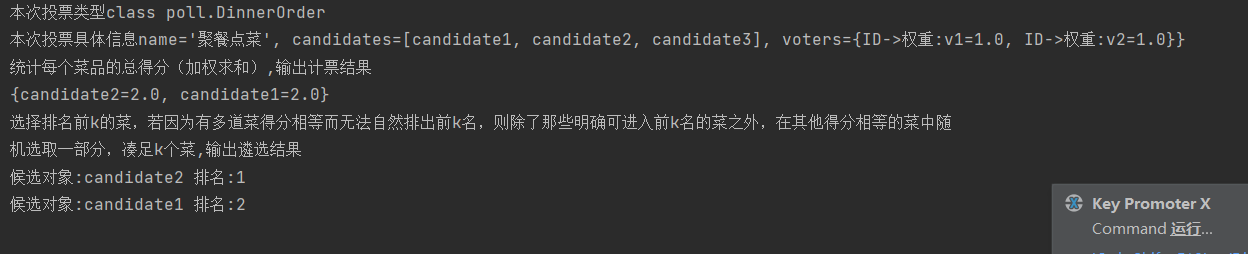
计票得到 candidate2->1

遴选得到candidate2>1

遴选得到candidate1>1

**测试结果：**





## 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。

# 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 |
| 2022.6.19 | 18.00-20.00 | 看懂任务指导书，然后开个头代码 | 提前完成 |
| 2022.6.20 | 9.45-12.00 | 把VoteType和VoteItem写完 | 未完成 |
| 2022.6.20 | 16.00-22.00 | 写完Vote，补充之前的测试样例，补完之前欠的 | 完成，并且超额完成 |
| 2022.6.21 | 8.00-11.30 | 根据Poll和GeneralPollImpl补充测试样例 | 完成，并且填补了一些实现方法 |
| 2022.6.21 | 17.30-21.30 | 计划补充完之前欠下的任务，还可以往后写 | 完成 |
| 2022.6.22 | 18.00-22.00 | 写三个应用场景 | 未完成，有点多 |
| 2022.6.23 | 18.50-22.50 | 写三个应用场景 | 大体上完成了 |
| 2022.6.24 | 12.00-22.00（中间恰个饭） | 为三个应用编写测试样例，及完善之前内容 | 大体上完成了 |
| 2022.6.25 | 12.00-23.00 | 写报告！写报告！写报告 | 未完成 |
| 2022.6.26 | 12.00-23.00 | 完善内容，以及任务14 | 必须得完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| Vote是否要重写Equals和hashCode  以及如何重写Equals和HahsCOde | 1. 首先我先重写，但是在后期进行样例测试之后发现始终有Vote加入不进去，调试后发现因为重写了Equals以及Votes存储容器是Set所以会自动去重 2. 则综上所述去除了重写 3. 重写这俩方法可以IDEA自动生成 |
| 如何POllTest测试样例编写，毫无头绪 | 1.以ElcetionAPP给定样例为模板，先完成ElectionAPP大体设计，然后复刻到ElectionTest，然后复刻到PollTest |
| 在没有Vote接口的情况下如何编写和使用Decorator设计模式 | 1. 自己编写一个VoteInterface接口，然后根据设计模式来编写VoteDecorator和RealNameVoteDecorator。 2. 在RealNameVoteAPPTest中提供了如何使用的思路和客户端使用类的代码示例 3. 如果后续要使用，可以重构Vote和VoteInterface |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训（必答）

1. 经验：
   1. 更进一步理解了设计模式，在亲自编写了Strategy和Visitor，Decorator之后了解到设计模式的好处以及强大之处
2. 教训：
   1. 在设计模式中，对于Decorator抽象类应该要实现Component接口，这样才符合Decorator设计模式

## 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？
   1. 差异：
      1. 面向ADT要考虑到更多内在的关系和联系，还需要具有一定的抽象思维能力，但是更加实用更加适用更多场景
      2. 面向应用场景编程,在编写难度上低于面向ADT，更加直接更加便于理解，但是适用场景单一而有限制，并不是一种较优的选择
      3. 体会到了复用的好处：
         1. 我们将Poll接口设计好，然后GeneralPollImpl设计好，这时候三个不同应用场景可以根据现有的基础上进行复用，特别方便
         2. 将共性提取好了之后，几个应用场景都可以使用，非常便捷。
         3. 委托，设计模式，代码的复用，可以让我们免除大量冗余的编程时间
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？
   1. 意义是：这些可以方便我们更好的理解ADT，以及在编程时候提醒自己哪里可能有问题哪里需要检查不变量，时刻注意表示泄露可以提高我们在使用时候的安全性，这是以后成熟的编写的要求
   2. 我愿意继续这么坚持编程，因为这是一个良好的编程习惯，并且有利于提高编写的程序的质量
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？
   1. 有，编写过JAVA飞机大战，JAVA城堡问题等
   2. 可以说别人提供的ADT和API用于自己的程序开发是把双刃剑
      1. 难处是：在使用之前需要理解看懂别人ADT/API的使用或者功能，如果没有理解好，则会出现一些错误在自己的程序开发，而且还很难发现啊！
      2. 好处是：很节省时间，用现成的API可以大大节省时间，并且一般第三方API大多时间复杂度已经优化过了
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？
   1. 语法驱动编程的功能非常强大，这一点在之前学习python时候已经感受到，比如之前自己写的py程序可以利用正则表达式通过python语言统计出三国演义中出现多少次“曹曰”统计次数。可以推而广之，语法驱动编程很强大
   2. 语法驱动编程难度也大，比如说在这次编写正则表达式，需要考虑的情况较多，所以正则表达式设计难度也有所提升
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？
   1. 我觉得可以完全搞定，但是需要更多的时间
   2. 我认为难度体现在：
      1. 首先是要从题目要求中抽象提取出几个关键事物
      2. 其次要挖掘出他们内在的联系
      3. 然后是设计ADT
      4. 还有一个难点是如何抽取出几个ADT的共性，利用共性编写好接口，然后子类与父类的关系，理解好异同才可以好好梳理出子类与父类的关系
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？
   1. 经验教训：
      1. 首先设计时候我们得自己先理解好规划好应该如何设计ADT，及每个ADT之间的联系和对应的AF，最后弄出个大致的UML然后再开始动手编程
      2. 应该先写接口或者抽象类，以此来规范我们后续的实现类
      3. 同理应该在此之后写测试样例，以此来约束我们的实现能够切合要求的spec和要求的功能
      4. 抽象，委派，设计模式比较难掌握，但是一旦掌握熟练运用之后，感觉变成能力有所上升
7. 关于本实验的工作量、难度、deadline。
   1. 工作量确实大，真不愧是压轴好戏，可以说lab3的任务量原超前两个，难度也有所提高，但是因为ddl是在考完试之后一周所以刚刚好！
   2. 我的评价是觉得这次试验ddl时间和对应的任务量安排的不错
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？
   1. 对于课程内容：我觉得学习CMU和MIT的这门课确实很有收获，但是如果是在对JAVA基础语法完全熟练之后，可以更容易上手这门课（但是我们没有先修课程java有些遗憾）。如果能够做了java项目才从头上课，会对里面的策略了解得更深
   2. 总而言之我觉得这门课让我对软件工程提及了极大的兴趣
   3. 对于任课老师：老师上课时候很负责很认真，答疑时候也非常不错，几次去答疑都很有耐心。