Team 2

Voting System

Software Design Document

Name (s): Bethany Freeman, Derrick Dischinger, Rock Zgutowicz

Date: (02/27/2024)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design details of the Voting System. The expected audience is the Teaching Staff for Software Engineering I, Spring 2024 at the University of Minnesota, Twin Cities. It will also serve as a reference document for members of team 2.

1.2 Scope

This document contains a complete description of the design of the Voting System. The scope of the software will be to quickly and accurately determine the winners of OPL and CPL elections. This has been created to reduce any inconsistencies with manually counting votes, and to promote a more fair election process.

1.3 Overview

This document describes the architecture and system design for Voting System, an election result determination system.

1.4 Definitions and Acronyms

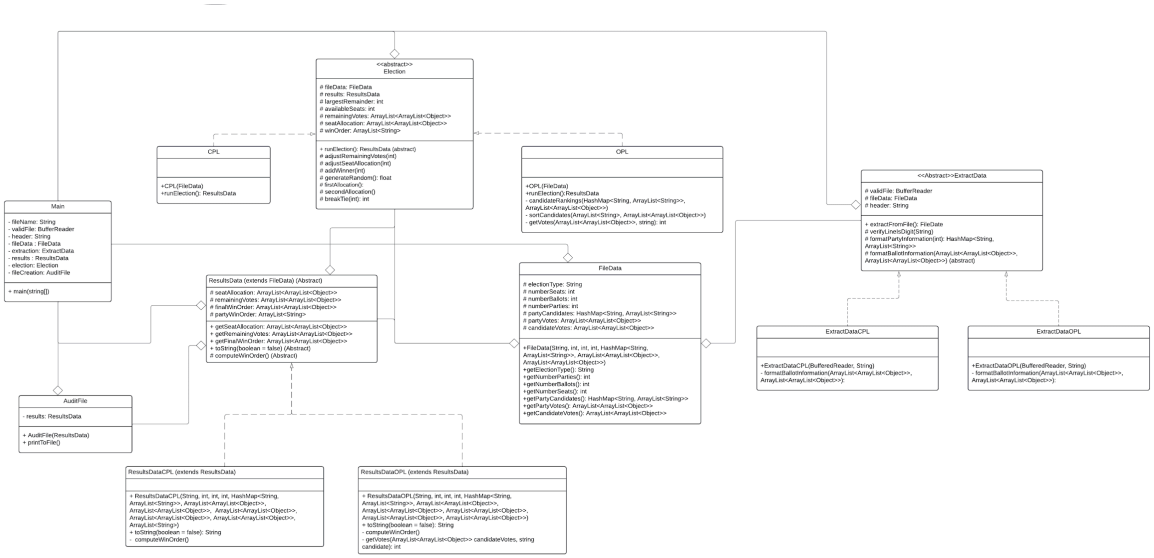
OPL - Open Party List
CPL - Closed Party List

2. SYSTEM OVERVIEW

The system will accept a formatted CSV file containing election results, through either the command line, or through a text prompt when run. The system will then verify the file, and perform calculations to determine the election results. Once this is done, an audit file will be generated, which election officials can use to announce the results of the election.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design



Main - Responsible for reading in files, creating and calling objects to extract file data, perform calculations on the data, and then producing an audit file. Main interacts with any class which inherits from ExtractData, so information for specific elections can be extracted. Main then interacts with any file which inherits from Election, so that a proper algorithm can be performed on the extracted data in order to produce results. With these results, Main interacts with the ResultsData and AuditFile classes in order to display information about the election to the user, as well as to generate the audit file.

AuditFile - Responsible for generating an audit file given the results of the extracted data. AuditFile interacts directly with the ResultsData class in order to produce the audit file. AuditFile is interacted with by Main in order for it to be passed the corresponding ResultsData object.

<<abstract>> Election - Provides the base information which every type of election should have. The OPL and CPL classes inherit from this class to provide further information relating to the specific election needs.

OPL - Performs computations specific to OPL elections. Inherits from Election to provide base information which is carried through each type of election. Main interacts with this class in order to pass a FileData object into it for it to perform calculations on in order to produce election results.

CPL - Performs computations specific to CPL elections. Inherits from Election to provide base information which is carried through each type of election. Main interacts with this class in order to pass a FileData object into it for it to perform calculations on in order to produce election results.

ResultsData - The ResultsData class is responsible for containing the final election results once the OPL or CPL algorithm has been run on the file data which has been extracted. ResultsData extends FileData in order to provide additional information needed in order to be used to display results or produce an audit file. Both the Main class as well as the AuditFile class interact with ResultsData in order to pass the necessary information from ResultsData into the AuditFile class in order to generate the audit file. Main interacts directly with ResultsData in order to display election results to the user.

FileData - Contains information regarding a specific CSV file. Any class which inherits from Election will be interacting with FileData as to perform any necessary calculations on the data found within it. Main interacts with this class to pass the information found within into these two classes; CPL and OPL. The same applies to any class which inherits the ExtractData class, as these classes store the extracted data within a FileData object. ResultsData extends this class in order to provide additional information needed to generate and display election results.

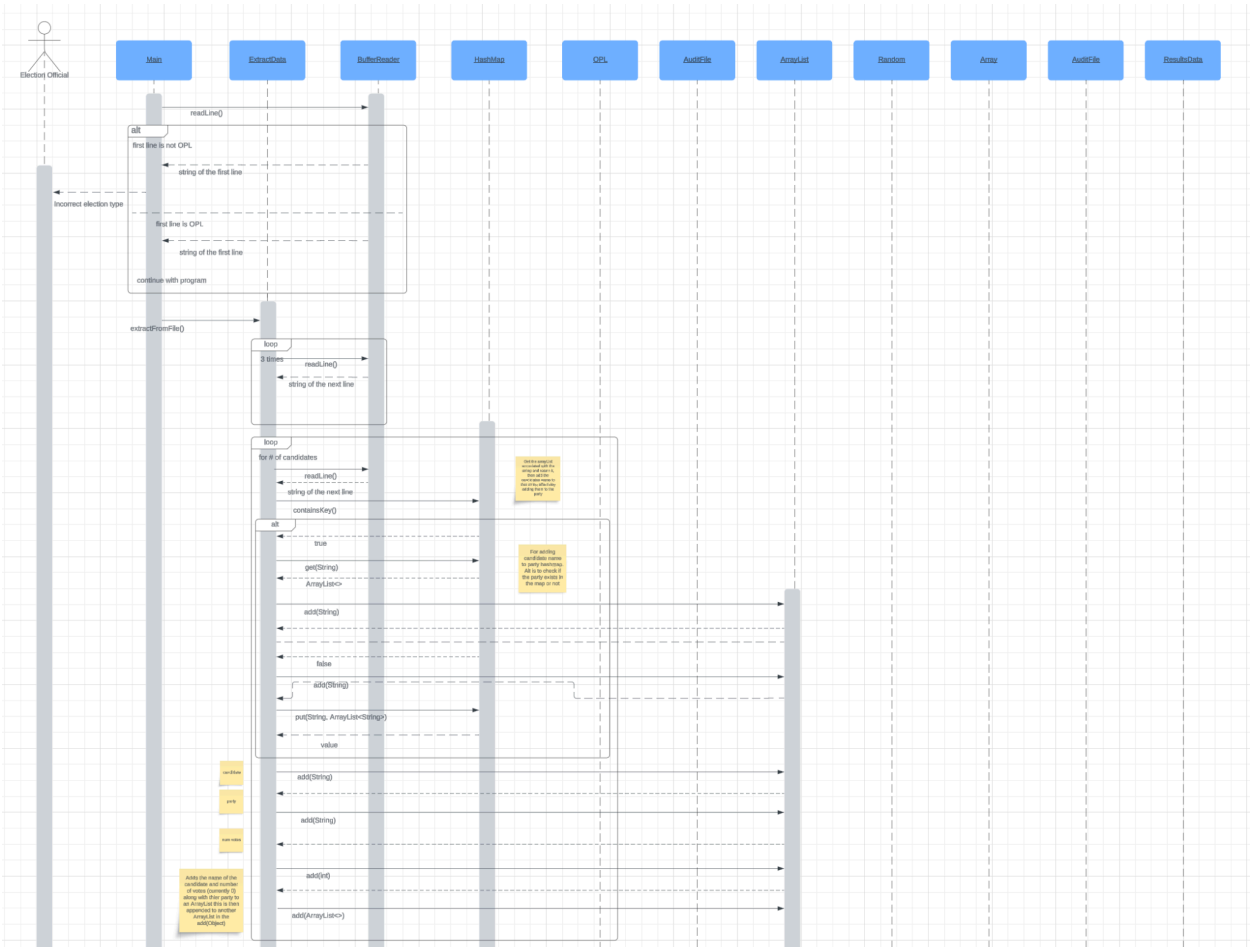
<<abstract> ExtractData - Provides the base information needed for extracting data out of any formatted CSV file. The ExctactDataCPL and ExtractDataOPL classes inherit from this class to provide further information relating to the specific election needs. ExtractData also interacts with FileData, as when data is extracted, the data is stored in a FileData object.

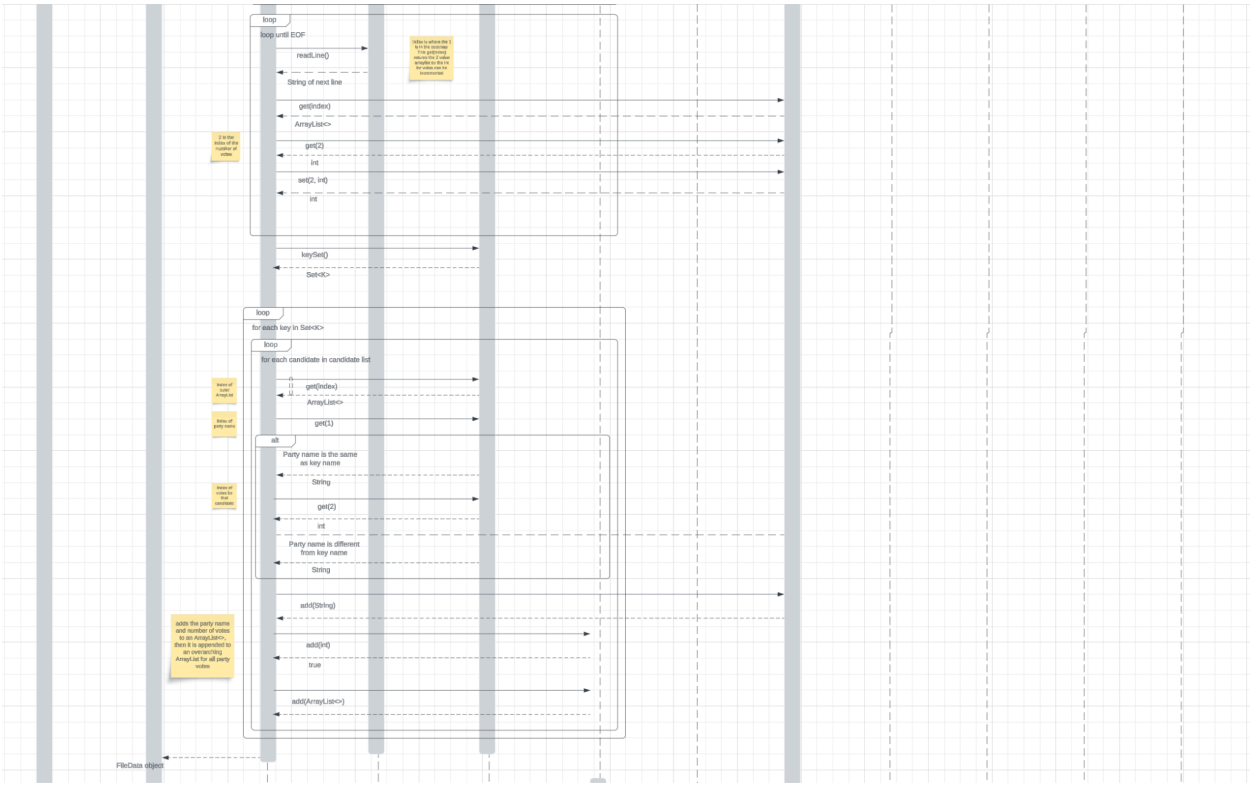
ExtractDataCPL - Extracts data from a CSV ballot file formatted for CPL elections. Inherits from Election to provide additional information needed to extract data from a CPL election file. This class is called by Main, which passes in the name of the file whose data must be extracted.

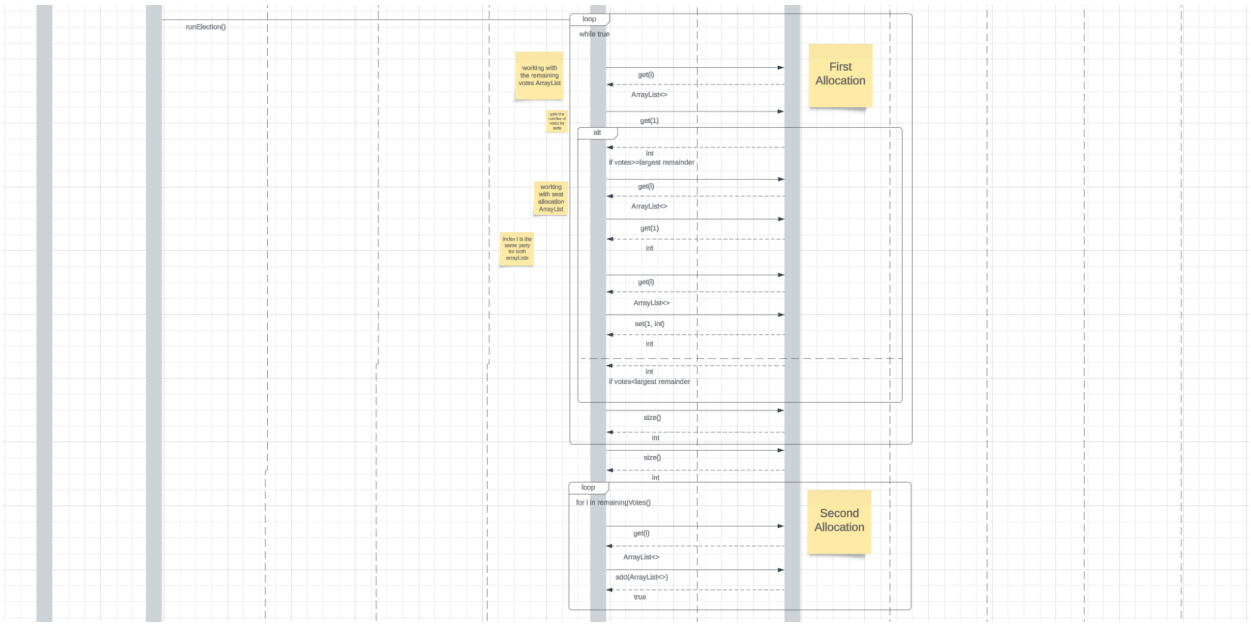
ExtractDataOPL - Extracts data from a CSV ballot file formatted for OPL elections. Inherits from Election to provide additional information needed to extract data from a OPL election file. This class is called by Main, which passes in the name of the file whose data must be extracted.

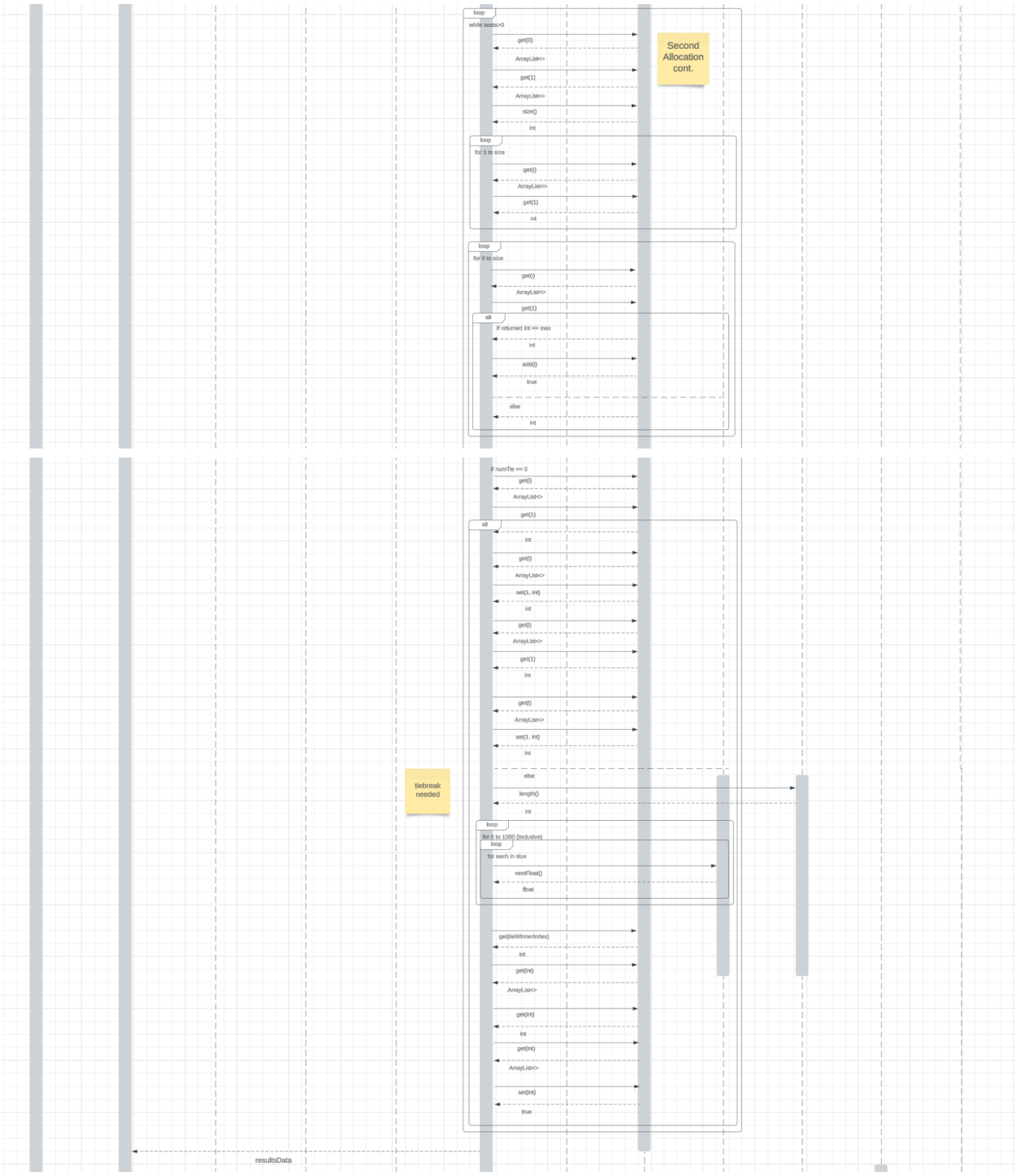
3.2 Decomposition Description

Sequence Diagram for an OPL election:

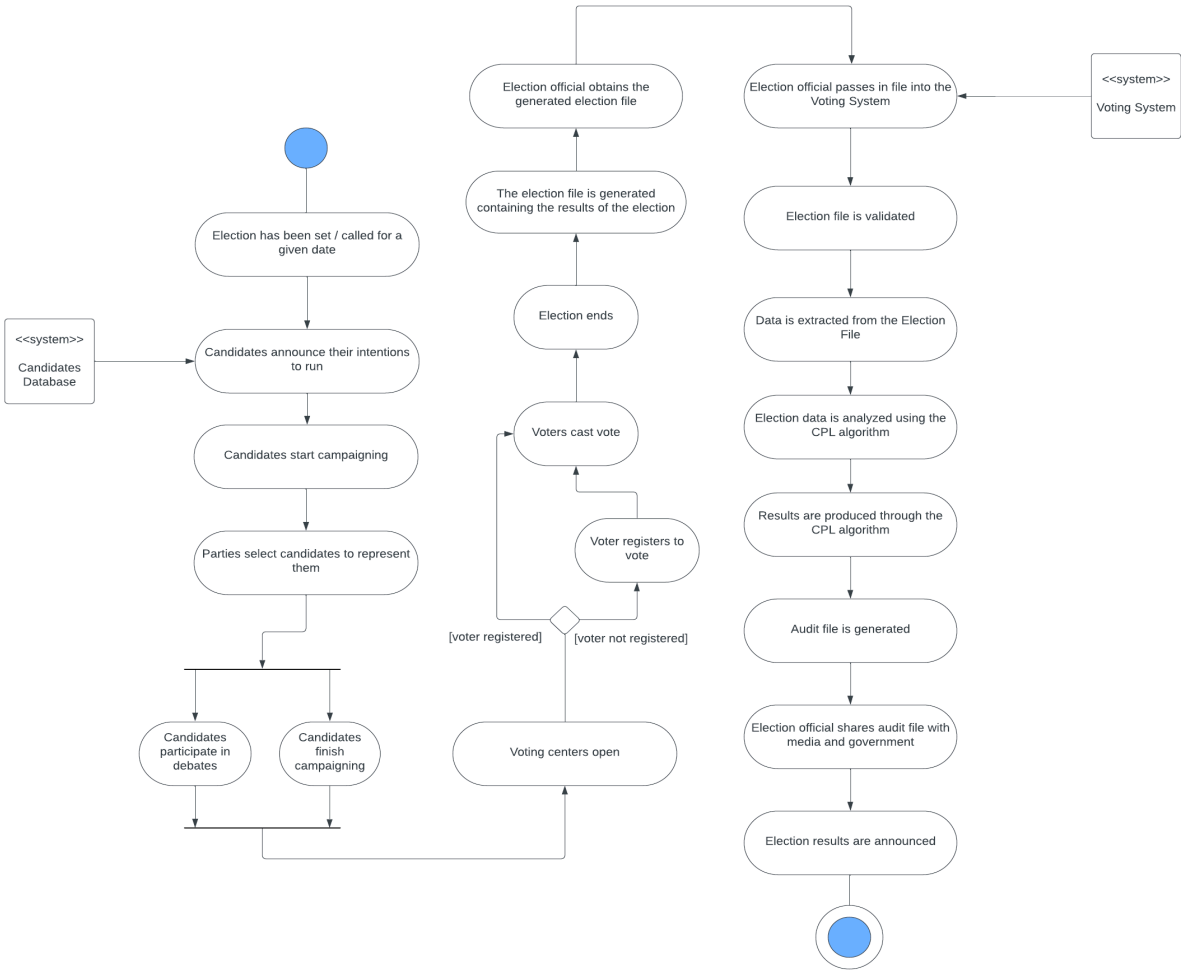



loop
loop until EOF
readLine()
String of next line
get(index)
ArrayList<>
get(2)
int
set(2, int)
int
keySet()
Set<K>
loop
for each key in Set<K>
loop
for each candidate in candidate list
get(index)
ArrayList<>
get(1)
alt
Party name is the same as key name
String
get(2)
int
Party name is different from key name
String
add(String)
add(int)
true
add(ArrayList<>)
FileData object


runElection()
loop
while true
working with the remaining votes ArrayList
get(i)
ArrayList<>
get(1)
alt
int
if votes>=largest remainder
get(i)
ArrayList<>
get(1)
int
get(i)
ArrayList<>
set(1, int)
int
int
if votes<largest remainder
size()
int
size()
int
First Allocation
loop
for i in remainingVotes()
get(i)
ArrayList<>
add(ArrayList<>)
true
Second Allocation


loop
while seats>0
get(0)
ArrayList<>
get(1)
ArrayList<>
size()
int
Second Allocation cont.
loop
for 1 to size
get(i)
ArrayList<>
get(1)
int
loop
for 0 to size
get(i)
ArrayList<>
get(1)
alt
If returned int == max
int
add(i)
true
else
int
If numTie == 0
get(i)
ArrayList<>
get(1)
alt
int
get(i)
ArrayList<>
set(1, int)
int
get(i)
ArrayList<>
get(1)
int
get(i)
ArrayList<>
set(1, int)
int
else
length()
int
tiebreak needed
loop
for 0 to 1000 (inclusive)
loop
for each in size
nextFloat()
float
get(tieWinnerIndex)
int
get(int)
ArrayList<>
get(int)
int
get(int)
ArrayList<>
set(int)
true
resultsData

Activity Diagram for a CPL election:



3.3 Design Rationale

Our system relies heavily on using abstract classes to define objects such as Election and ExtractData. Java's polymorphic nature allowed us to create individual classes that implemented the abstract classes while still allowing them to be treated as one object (CPL can be treated like an election throughout the system). This abstraction allows us to extract common features between different methods of voting while leaving the system modular and extensible to the possibility of implementing new voting types.

We decided to use nested ArrayLists for the majority of our data storage, while they were a little more complex than our initial plan of using HashMaps, they allowed us to keep the data in a sorted order which was critical while counting votes. We opted to not use classes as the data structure for storing party and candidate information as while the designs we came up with could be easily used for one election type they did not generalize very well to the other. The biggest drawback to using nested ArrayLists is the complexity of accessing and sorting the information, we determined that this was not a big enough drawback to use classes instead.

4. DATA DESIGN

4.1 Data Description

While extracting data from any input files, the data is then held in a ‘FileData’ object. This class contains information regarding the CSV file, such as the election type, number of seats, ballots, parties, candidates, and votes.

We then have classes which inherit from the abstract Election class, which contain algorithms used in order to determine the election results. The election class contains information pertaining to any election, such as seat allocation, record data, results, and more. Any class that inherits from Election must provide additional information needed in order to run their respective algorithms, such as any additional data structures, such as a hashmap and arraylist, seen in the class OPL.

Once the respective algorithm has been run on the ‘FileData’ object, a new object which extends from the ‘FileData’ object is created, ‘ResultsData’. This object contains information necessary to output the results of the election to the user, as well as to generate the audit file. The generation of the audit file is done through the ‘AuditFile’ class, which uses the generated ‘ResultsData’ in order to create the file.

The ‘Main’ class is responsible for reading in the input file, as well as making any necessary calls to classes to perform calculations, and to generate the audit file.

4.2 Data Dictionary

AuditFile

Attributes:
results: ResultsData

Methods:
printToFile()

CPL

Methods:
runElection(): ResultsData

<<abstract>> Election

Attributes:
fileData: FileData
results: ResultsData
largestRemainder: int
remainingVotes: ArrayList<ArrayList<String, int>>
availableSeats: int
winOrder: ArrayList<ArrayList<String, String, int, int>>
seatAllocation: arrayList<ArrayList<Object>>

Methods:
runElection(): ResultsData (abstract)
adjustRemainingVotes(int)
adjustSeatAllocation(int)
firstAllocation()
secondAllocation()
breakTie(int, ArrayList): int
generateRandom(): float
addWinner(int)

<<abstract> ExtractData

Attributes:
validFile: BufferReader
data: FileData
header: String

Methods:
extractFromFile(): FileDate
VerifyLineIsDigit(String)
formatPartyInformation(int): HashMap<String, ArrayList<String>>
formatBallotinformation(ArrayList<ArrayList<String,int>>,ArrayList<ArrayList<String, int>>):

ExtractDataCPL

Methods:

formatBallotinformation(ArrayList<ArrayList<String,int>>,ArrayList<ArrayList<String, int>>):

ExtractDataOPL

Methods:

formatBallotinformation(ArrayList<ArrayList<String,int>>,ArrayList<ArrayList<String, int>>):

FileData

Attributes:

electionType: String
numberSeats: int
numberBallots: int
numberParties: int
partyCandidates: HashMap<String, ArrayList<String>>
partyVotes: ArrayList<ArrayList<String, int>>
candidateVotes: ArrayList<ArrayList<String, int>>

Methods:

getElectionType(): String
getNumberParties(): int
getNumberBallots(): int
getNumberSeats(): int
getPartyCandidates(): HashMap<String, ArrayList<String>>
getPartyVotes(): ArrayList<ArrayList<String, int>>
getCandidateVotes(): ArrayList<ArrayList<String, int>>

Main

Attributes:
fileName: String
validFile: BufferReader
header: String
fileData : FileData
extraction: ExtractData
results: ResultsData
election: Election
fileCreation: AuditFile

Methods:
main(string[])

OPL

Methods:
candidateRankings(HashMap<String, ArrayList<String>>, ArrayList<ArrayList<Object>>)
sortCandidates(ArrayList<String>, ArrayList<ArrayList<Object>>)
getVotes(ArrayList<ArrayList<Object>>, string): int

ResultsData

Attributes:
seatAllocation: ArrayList<ArrayList<String, int[]>>
remainingVotes: ArrayList<ArrayList<String, int>>
candidateRanking: HashMap<String, ArrayList<String>>
finalWinOrder: ArrayList<ArrayList<String, String, int, int>>

Methods:
getSeatAllocation: ArrayList<ArrayList<Object>>
getRemainingVotes: ArrayList<ArrayList<Object>>
getFinalWinOrder: ArrayList<ArrayList<Object>>
toString(boolean = false) (Abstract)
computeWinOrder() (Abstract)

5. COMPONENT DESIGN

AuditFile

AuditFile(ResultData results)

results = results

PrintToFile()

create a new file to output to
printLine = this.results.toString()
print to the file printLine
return

CPL

CPL(FileData fileData)

fileData = fileData
availableSeats = fileData.getNumberSeats()
finish creating remaingVotes
finish creating seatAllocation
finish creating finalWinOrder
// largestRemainder calculation
ballots = fileData.getNumberBallots()
largestRemainder = ballots/availableSeats

runElection()
firstAllocation()
secondAllocation()
finish creating results
return results

Election

abstract runElection()

adjustRemainingVotes(int index)

```
value = remainingVotes.get(index).get(1)
value -= largestRemainder
remainingVotes.get(index).set(1, value)
return
```

adjustSeatAllocation(int index)

```
value = seatAllocation.get(index).get(1)
value += 1
seatAllocation.get(index).set(1, value)
return
```

addWinner(int index)

```
winner = seatAllocation.get(index.get(0)
finalWinOrder.add(winner)
return
```

generateRandom()

```
for 0 to 1000
        r = random.nextfloat() * (10-1)
return r
```

firstAllocation()

```
while true
        votes = remaingVotes.get(i).get(1)
        if votes >= largestRemainder
                adjustRemaingVotes(i)
                adjustSeatAllocation(i)'
                addWinner(i)
                avaliableSeats--
        else
                underRemain++

        i++

        if availableSeats <= 0 //Are there no seats left?
                break
        if underRemain >= remainingVotes.size() // Are all of the parties under the
                                                // remainder?
```

```
                    break
                if i >= remaingVotes.size() // start round robin over
                    underRemain = 0
                    i = 0

secondAllocation()
        // creates a new ArrayList<ArrayList<Object>> which is a copy of remainingVote
        // for remainingVotes.size()
            innerList = remaingVotes.get(i)
            remainVotesNew.add(innerList)

        while availableSeats > 0
            max = remainVotesNew.get(0).get(1)
            index = 0

            for 1 to remainVotesNew.size()
                current = remainVotesNew.get(i).get(1)

                if current > max
                    max = current
                    index = i

            for remainVotesNew.size()
                // We don't need to check if the same index of our max is equal to
                // our max
                if i == index
                    continue

                current = remainVotesNew.get(i).get(1)
                if current == max
                    numTie += 1 // This is how many ties there are
                    indexTie.add(i) //This is an arrayList of indices tied to what
                                    // locations have ties

            if numTie != 0
                location = breakTie(int numTie)
                index = indexTie.get(location)
            adjustSeatAllocation(index)
            value = remainVotesNew.get(index).get(1)
            value = value - largestRemainder
            remainVotesNew.get(index).set(1, value)

        return
```

breakTie(int numTie)

```
compVal = generateRandom()
create new Array float[] of size numTie called tieBreak

for tieBreak.length
        tieBreak[i] = generateRandom()

// make the current smallest value the absolute value of the difference between
// compVal and the first value in tieBreak
curSmallest = |compVal - tieBreak[0]|
index = 0
isSame = false

//I will be putting |dif| from now on, this stands for |compVal - tieBreak[i]|
for 1 to tieBreak.length
        if |dif| < curSmallest
                curSmallest = |dif|
                index = i
                isSame = false
        else if |dif| == curSmallest
                isSame = true

if isSame
        breakTie(numTie)

return i
```

OPL

OPL(FileData fileData)

```
fileData = fileData
availableSeats = fileData.getNumberSeats()
finish creating remaingVotes
finish creating seatAllocation
finish creating finalWinOrder
// largestRemainder calculation
ballots = fileData.getNumberBallots()
largestRemainder = ballots/availableSeats
```

runElection()

```
    firstAllocation()
    secondAllocation()
    candidateRankings(HashMap<string,ArrayList<String>>
    fileData.getPartyCandidates, ArrayList<ArrayList<Object>>
    fileData.getCandidateVotes)
    finish creating results
    return results
```

candidateRankings(HashMap<string,ArrayList<String>>

```
    fileData.getPartyCandidates,ArrayList<ArrayList<Object>>
    fileData.getCandidateVotes)
    for party in partyCandidates
            candidates = partyCandidates.get(party)
            // Sort candidates based on number of votes
            sortCandidates(candidates, candidateVotes)
            // Update partyCandidates with the sortedCandidate ArrayList
            partyCandidates.get(party).set(1, candidates)
  return
```

sortCandidates(ArrayList<String> candidates, ArrayList<ArrayList<Object>> candidateVotes)

```
    // Define custom comparator function to sort candidates based on number of votes
    @override
    comparator = function(candidate1, candidate2)
            votesCandidate1 = getVotes(candidateVotes, candidate1)
            votesCandidate2 = getVotes(candidateVotes, candidate2)
            value = votesCandidate2 - votesCandidate1
            return value

    //call custom comparator with sort
    sort(candidates, comparator)
```

getVotes(ArrayList<ArrayList<Object>> candidateVotes, string candidate)

```
    for entry in candidateVotes
            if entry.get(0) == candidate
                    return entry.get(1)
```

ExtractData

extractFromFile():

```
line = validFile.readLine()
verifyLineIsDigit(line)
numSeats = Integer.parseInt(line)

line = validFile.readLine()
verifyLineIsDigit(line)
numBallots = Integer.parseInt(line)

line = validFile.readLine()
verifyLineIsDigit(line)
numParties = Integer.parseInt(line)

create new HashMap<String, ArrayList<String>> called partyCandidates
// Both ArrayLists will present as tuples of the form <<String, int>>
create new ArrayList<ArrayList<Object>> called partyVotes
create new ArrayList<ArrayList<Object>> called candidateVotes
partyCandidates=formatPartyInformation(numParties,partyVotes, candidateVotes)

formatBallotInformation(partyVotes, candidateVotes)
fileData = FileData(header, numSeats, numBallots, numParties, partyCandidates,
partyVotes, candidateVotes)
```

verifyLineIsDigit(String line)

```
for line.length()
        if char of line at i is not a digit
                send a message to the user
                exit the system
return
```

formatPartyInformation(int numParties,ArrayList<ArrayList<Object>> partyVotes,ArrayList<ArrayList<Object>> candidateVotes parameter)

```
create empty HashMap named partyCandidates
for numParties
        line = validFile.readLine()
        party = line.split(",")
        partyName = party[0].trim()
        create new ArrayList called partyInner
        partyInner.add(partyName)
        partyInner.add(0)
        partyVotes.add(partyInner)
        create new ArrayList candidates
        for 1 to party.length
                candidates.add[i].trim()
                create new ArrayList called candidateInner
                candidateInner.add(i)
                candidateInner.add(0)
                candidateVotes.add(candidateInner)
        partyCandidates.put(partyName, candidates)
return partyCandidates
```

abstract formatBallotInformation(ArrayList<ArrayList<Object>> partyVotes, ArrayList<ArrayList<Object>> candidateVotes)

ExtractDataCPL

ExtractDataCPL(BufferedReader validFile, String header)

```
validFile = validFile
header = header
```

formatBallotInformation(ArrayList<ArrayList<String,int>>partyVotes, ArrayList<ArrayList<String, int>> candidateVotes)

```
while not end of file
        line = validFile.readLine()
        for character in line
                if character is 1
                        index = i
                        break
        count = partyVotes.get(index).get(1)
        count += 1
        partyVotes.get(index).set(1, count)
return
```

ExtractDataOPL

ExtractDataOPL(BufferedReader validFile, String header)

```
validFile = validFile
header = header
```

formatBallotInformation(ArrayList<ArrayList<String,int>>partyVotes, ArrayList<ArrayList<String, int>> candidateVotes)

```
while not end of file
        line = validFile.readLine()
        for character in line
                if character is 1
                        index = i
                        break
        count = candidateVotes.get(index).get(1)
        count += 1
        candidateVotes.get(index).set(1, count)
return
```

FileData

getElectionType()

```
return electionType
```

getNumberParties()

```
return numberParties
```

getNumberBallots()

```
return numberBallots
```

getNumberSeats()

```
return numberSeats
```

getPartyCandidates()

```
return partyCandidates
```

getPartyVotes()

```
return partyVotes
```

getCandidateVotes()

```
return candidateVotes
```

ResultsData

getSeatAllocation()
return seatAllocation

getRemainingVotes()
return remainingVotes

getFinalWinOrder()
return finalWinOrder

abstract toString()

abstract computeWinOrder()

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

For testers and developers, the program will be executed along with the formatted CSV file name containing the results obtained from the ballots as an input. For normal users, or election officials, when running the program, they will be prompted to enter the file name of the formatted CSV file containing the results obtained from the ballots. For both types of users, they can expect to view output from the program displaying the winning parties, seat winners, the number of seats won and the number of votes for each winner. Users will also see where the generated audit file has been saved, which will contain additional information about the results. Some of the additional information will include the type of election, the number of parties, the number of ballots cast, the number of seats available, all candidates running with what party they represent,as well as the stats for each party, such as number of votes, first allocation of seats, remaining votes, second allocation of seats, final seat total, and percentage of votes to percentage of seats.

Along with this, users can expect the input file to be validated to ensure that the file has been formatted correctly. If the file cannot be validated, the users will expect to receive any error information regarding what is wrong with the file.

If any other errors occur in the process of obtaining election results, users can expect to see what kind of error occurred, along with steps on how they may try to correct the error.

6.2 Screen Images

Example input for normal user:

```
$> java VotingSystem

Please enter the name of the ballot file: ballot.CSV
```

Example input for tester / developers:

```
$> java VotingSystem ballot.CSV
```

Example error output (could not validate file):

```
Error: Invalid file. Input file must be a valid CSV, and follow proper
formatting guidelines.
```

Example Output to User for OLP Election:

```
-----------------------------------------------------------------------
     Winning     |      Seat       |     Seat     |      Number
     Parties     |     Winners     |      Won     |     Of Votes
-----------------------------------------------------------------------
    Republican   |      Allen      |       1      |      23,000
    Democratic   |      Joe        |       2      |      12,000
    Republican   |      Nikki      |       3      |      10,000
-----------------------------------------------------------------------
Audit File saved: C:\Users\freem627\Documents\AuditFiles\OPL_Election_Results_2024-02-12_10:38:47
```

Example Output to Audit File for OLP Election:

```
OPL Election
4 Parties
60,000 Ballots Cast
3 Seats Avaliable
-----------------------------------------------
    Party     |     Candidates
-----------------------------------------------
  Republican  |      Allen, Nikki
  Democratic  |      Joe, Laura
  Green       |      Rob
  Independent |      Henry
-----------------------------------------------

-------------------------------------------------------------------------------------------------------
    Parties   |    Votes    |    First     |    Remaining    |    Second    |    Final    |    % of Vote
              |             |  Allocation  |      Votes      |  Allocation  |    Seat     |       to
              |             |  Of Seats    |                 |  Of Seats    |    Total    |    % of Seats
-------------------------------------------------------------------------------------------------------
   Republican |    32,000   |       1      |      12,000     |       1      |      2      |     53%/66%
   Democratic |    23,000   |       1      |       3,000     |       0      |      1      |     38%/34%
   Green      |     3,000   |       0      |       3,000     |       0      |      0      |      5%/0%
   Independent|     2,000   |       0      |       2,000     |       0      |      0      |      3%/0%
-------------------------------------------------------------------------------------------------------

-----------------------------------------------------
    Winning     |      Seat       |     Seat     |      Number
    Parties     |     Winners     |      Won     |     Of Votes
-----------------------------------------------------------------
   Republican   |      Allen      |       1      |      23,000
   Democratic   |      Joe        |       2      |      12,000
   Republican   |      Nikki      |       3      |      10,000
-----------------------------------------------------------------
```

6.3 Screen Objects and Actions

In the example output photos, '$>' represents the command line where the user has control over the terminal.

7. REQUIREMENTS MATRIX

VF_001	Main
VF_002	Main
EI_001	ExtractData, ExtractDataCPL, ExtractDataOPL, FileData
VA_001	Election, CPL, ResultsData
VA_002	Election, OPL, ResultsData
TB_001	Election
AF_001	ResultsData, AuditFile
DR_001	ResultsData, Main