Team 16

Trusty Vote Counter

Software Design Document

Names: Alex Iliarski, Ethan Loukusa, Gideon Tan, Robert Wang

Date: 03/02/2023

Table of Contents

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of *Trusty Vote Counter* (TVC). Additionally, this document will provide clear charts depicting the different objects and methods being used in each use case and the relationship between different classes and objects. This document is for use by election officials, developers, and testers.

1.2 Scope

This document contains a complete description of TVC, a Java program designed to determine election results for a given election using either the Instant Runoff (IR) voting system or the Closed Party List (CPL) voting system. TVC is a largely beneficial program in that it automates an otherwise arduous and labor-intensive task in determining election results, as well as eliminating errors in the counting process. TVC is intended to be used in conjunction with or as part of an online interactive voting system, but can be used independently as well.

1.3 Overview

This document continues in section 2 by giving a brief overview of *Trusty Vote Counter*.

Section 3 describes the system architecture of TVC, in particular the architectural design, the decomposition description, and design rationale.

Section 4 describes the data design. This section outlines all of the classes and their specific methods used in TCV.

Section 5 describes the component design. This section describes each object function more comprehensively, including algorithms and pseudo code.

Section 6 describes the human interface design.

Lastly, to connect back to our software requirements specification document, section 7 shows how each part of the design relates to the requirements set in place for TVC.

1.4 Reference Material

[IEEE] The applicable IEEE standards are published in "IEEE Standards Collection", 2001 edition.

The principal source of textbook material is "Software Engineering" by Ian Sommerville(Pearson

2015).

Reaves, Michael J. “Software Design Description Jacksonville State University Computing and Information Sciences Web Accessible Alumni Database.”Jacksonville State University, 2003.

The Software Requirements Specification by Robert Wang, Alex Iliarski, Ethan Loukusa, Gideon Tan https://docs.google.com/document/d/1W9HJ30_GzC4F3dRuatKkImuk7_hfZnDh/edit?usp=sharing&ouid=117208831313787612273&rtpof=true&sd=true

1.5 Definitions and Acronyms

Term	**Definition**
TVC	Stands for *Trusty Vote Counter*, the product name
CPL	Acronym for “Closed Party List” voting system, which is a voting system in which each political party has pre-decided who will receive the seats allocated to that party in the elections, the winner will be decided based on these orderings. Voters vote for a preferred party, and the percentage of votes that each party receives, along with the greatest remainder algorithm, determines the number of seats allocated to each party.
CSV	Acronym for “Comma Separated Values”, which is a file format where different columns of a table are represented as a list of values with commas in between each value and the rows represent the different rows in such a table.
Git	Git is a “version control” software that is used to help manage the changes inside a document that helps a team of developers keep track of any changes and work on the same file effectively.
Github	Github is a website that uses “Git” to store “repositories” which contain the programs that developers are using, and allows the team to maintain a main file where production-ready code is stored
IDE	Acronym for “Integrated Development Environment”, which is the software that is used to write, compile, and or interpret the code that developers write.
IR	Acronym for “Instant Runoff” voting system, which is a type of ranked preferential voting method. It uses a majority voting rule in single-winner elections where there are more than two candidates.

Terminal	A terminal is simply a text-based interface to the computer. In a terminal, you can type commands, manipulate files, execute programs, and open documents.
Version Control	This refers to the ability to keep track of changes to a certain document and allows ease of access to past versions of documents.
UML	Acronym for Unified Modeling Language, which is a pictorial representation of what the structure of code should look like without using too much coding language specific terminology

2. System Overview

TVC is used to handle tabulation of ballots and determining winners in an election. The system can handle Instant Runoff and Closed Party List elections. Given an input file with all the collected ballots from the election, the system will automatically output a winner to the screen and create an audit file with specific data from the election. This ensures that there is no manual intervention necessary to determine a winner, ensuring correct results and limiting opportunities for fraud or user error. This allows for a quick and secure ballot counting process, ensuring that voters have a greater trust in the accuracy of the election results. The product will be modularly designed to be reusable for different kinds of elections and ballot types.

The long term goal for TVC is to be used as part of an online integrated voting system, which will complete the election process from start to finish. TVC does not handle collecting ballots, security at the voting place, etc. TVC assumes an input file to be correctly formatted and to contain no mistakes, as that will be handled beforehand.

3. System Architecture

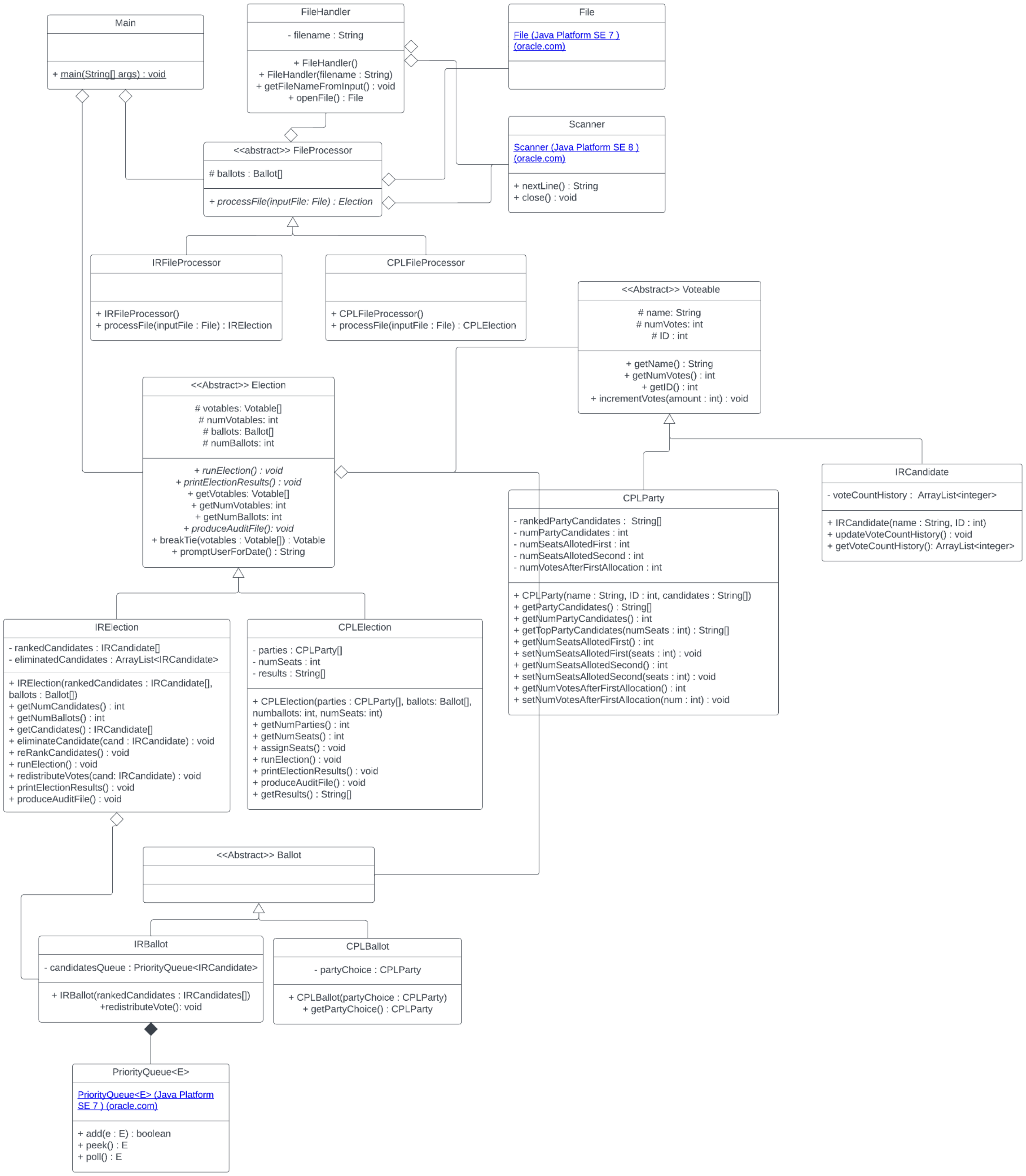
3.1 Architectural Design

The main modules that will be contained in the architectural design of this project are the main file, file handler, file processor, voteable candidate, ballot classes, and the election. The usage of each class in the system will be in the above order from the input of the file to the main class to the outcome of the election.

First, the user will be running the main class to begin using Trusty Vote Counter, and will either input a file name along with the call to main or not. The file handler will either take in the file name from input, or if there is not a file given, will prompt the user for the ballot file and will send the data to the file processor. The file processor is an abstract class that has the IR file processor and the CPL file processor classes as its children, which will handle the file according to its election type.

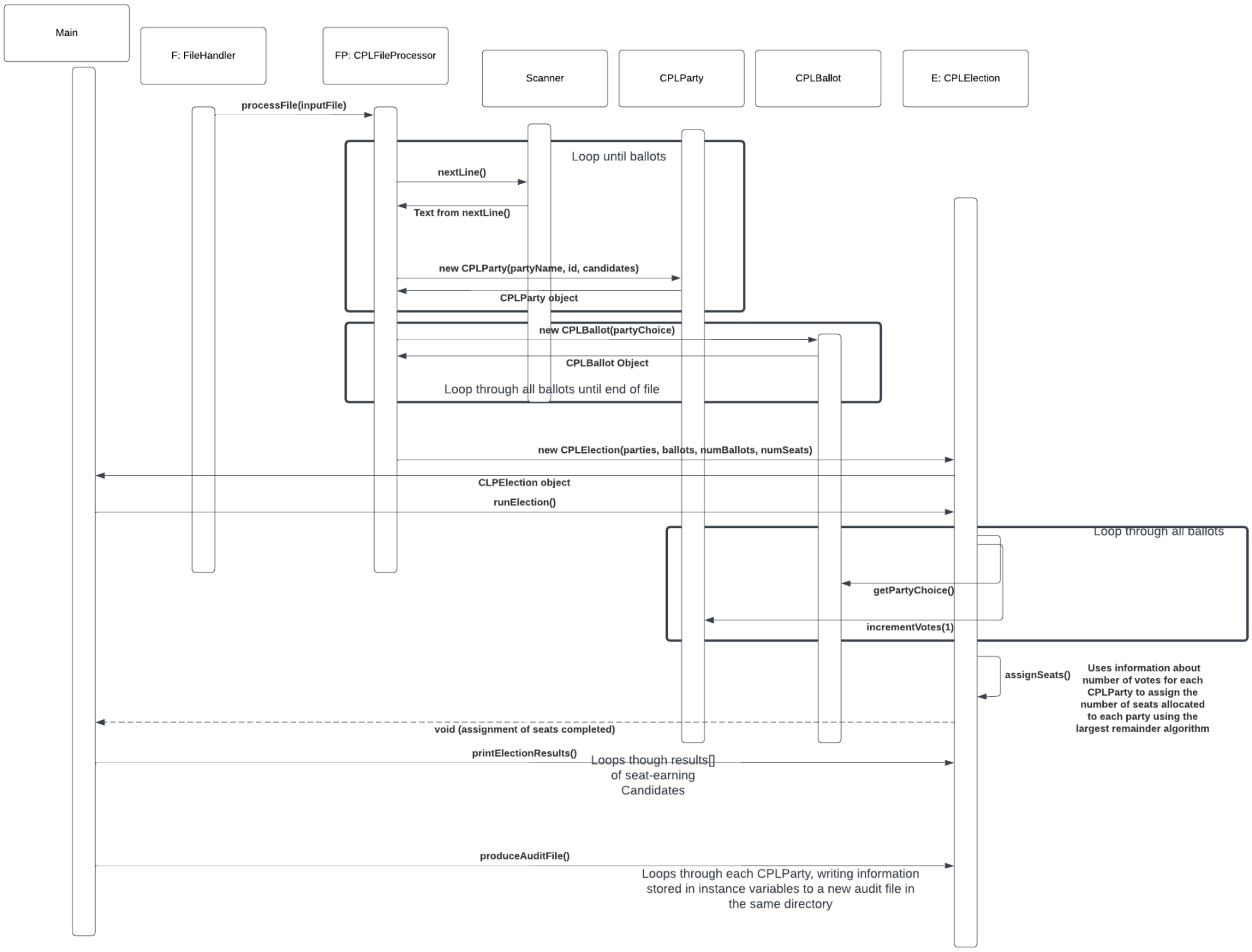
The file processor classes will be used with the ballot class, which includes the CPL Ballot and IR Ballot classes. The file processor will format information extracted from the input file into a ballot object, which consists of individual candidate objects and their rankings. These ballots, along with information about candidates, make up an election object. This election can then be run using a runElection method within the election object. This runs the election, and outputs the results to the terminal.

The UML class diagram which depicts the architectural design of TVC is shown on the next page. A bigger version can be found in the Github repository linked in Section 8.


Main
+ main(String[] args) : void
FileHandler
- filename : String
+ FileHandler()
+ FileHandler(filename : String)
+ getFileNameFromInput() : void
+ openFile() : File
File
File (Java Platform SE 7) (oracle.com)
Scanner
Scanner (Java Platform SE 8) (oracle.com)
+ nextLine() : String
+ close() : void
<<abstract>> FileProcessor
# ballots : Ballot[]
+ processFile(inputFile: File) : Election
IRFileProcessor
+ IRFileProcessor()
+ processFile(inputFile : File) : IRElection
CPLFileProcessor
+ CPLFileProcessor()
+ processFile(inputFile : File) : CPLElection
<<Abstract>> Voteable
# name: String
# numVotes: int
# ID : int
+ getName() : String
+ getNumVotes() : int
+ getID() : int
+ incrementVotes(amount : int) : void
<<Abstract>> Election
# votables: Votable[]
# numVotables: int
# ballots: Ballot[]
# numBallots: int
+ runElection() : void
+ printElectionResults() : void
+ getVotables: Votable[]
+ getNumVotables: int
+ getNumBallots: int
+ produceAuditFile(): void
+ breakTie(votables : Votable[]) : Votable
+ promptUserForDate() : String
IRCandidate
- voteCountHistory : ArrayList<integer>
+ IRCandidate(name : String, ID : int)
+ updateVoteCountHistory() : void
+ getVoteCountHistory(): ArrayList<integer>
CPLParty
- rankedPartyCandidates : String[]
- numPartyCandidates : int
- numSeatsAllotedFirst : int
- numSeatsAllotedSecond : int
- numVotesAfterFirstAllocation : int
+ CPLParty(name : String, ID : int, candidates : String[])
+ getPartyCandidates() : String[]
+ getNumPartyCandidates() : int
+ getTopPartyCandidates(numSeats : int) : String[]
+ getNumSeatsAllotedFirst() : int
+ setNumSeatsAllotedFirst(seats : int) : void
+ getNumSeatsAllotedSecond() : int
+ setNumSeatsAllotedSecond(seats : int) : void
+ getNumVotesAfterFirstAllocation() : int
+ setNumVotesAfterFirstAllocation(num : int) : void
IRElection
- rankedCandidates : IRCandidate[]
- eliminatedCandidates : ArrayList<IRCandidate>
+ IRElection(rankedCandidates : IRCandidate[], ballots : Ballot[])
+ getNumCandidates() : int
+ getNumBallots() : int
+ getCandidates() : IRCandidate[]
+ eliminateCandidate(cand : IRCandidate) : void
+ reRankCandidates() : void
+ runElection() : void
+ redistributeVotes(cand: IRCandidate) : void
+ printElectionResults() : void
+ produceAuditFile() : void
CPLElection
- parties : CPLParty[]
- numSeats : int
- results : String[]
+ CPLElection(parties : CPLParty[], ballots: Ballot[], numballots: int, numSeats: int)
+ getNumParties() : int
+ getNumSeats() : int
+ assignSeats() : void
+ runElection() : void
+ printElectionResults() : void
+ produceAuditFile() : void
+ getResults() : String[]
<<Abstract>> Ballot
IRBallot
- candidatesQueue : PriorityQueue<IRCandidate>
+ IRBallot(rankedCandidates : IRCandidates[])
+redistributeVote(): void
CPLBallot
- partyChoice : CPLParty
+ CPLBallot(partyChoice : CPLParty)
+ getPartyChoice() : CPLParty
PriorityQueue<E>
PriorityQueue<E> (Java Platform SE 7) (oracle.com)
+ add(e : E) : boolean
+ peek() : E
+ poll() : E

3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object oriented description. For a functional description, put top level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.


Main
F: FileHandler
FP: CPLFileProcessor
Scanner
CPLParty
CPLBallot
E: CPLElection
processFile(inputFile)
Loop until ballots
nextLine()
Text from nextLine()
new CPLParty(partyName, id, candidates)
CPLParty object
new CPLBallot(partyChoice)
CPLBallot Object
Loop through all ballots until end of file
new CPLElection(parties, ballots, numBallots, numSeats)
CLPElection object
runElection()
Loop through all ballots
getPartyChoice()
incrementVotes(1)
assignSeats()
Uses information about number of votes for each CPLParty to assign the number of seats allocated to each party using the largest remainder algorithm
void (assignment of seats completed)
printElectionResults()
Loops though results[] of seat-earning Candidates
produceAuditFile()
Loops through each CPLParty, writing information stored in instance variables to a new audit file in the same directory

The sequence diagram for the CPL election process is shown above.

An activity diagram for the IR election can be found further below in the SDD.

3.3 Design Rationale

Trusty Vote Counter is built using Java, which is an Object Oriented Programming language, making it very simple to create customer objects and classes to make the program modular in design and very extensible. C/C++ was also considered to be the language of choice in building Trusty Vote Counter, however the architecture of C, which is procedural, does not match with the modular outlook of the problem that this product is trying to solve. C++ on the other hand is also object oriented, but is less portable because C++ is platform reliant and needs to be compiled on every platform. Java, on the other hand, is both a compiled and interpreted language which is not platform dependent so it can be used more easily by users of all types.

4. Data Design

4.1 Data Description

The input data for this system will be stored in its separate file in a Comma Separated Values (CSV) format, which will be read by the file handler classes that we have created. For the voting process where order is very important, this product will be implementing lists as a way to organize and store the data about candidates and parties in both types of elections. Each node in the queue will be a class object of an IR candidate or a Party name, and each node will store its own important information like vote count and functions.

4.2 Data Dictionary

<<abstract>> Ballot

CPLBallot:

- Attributes:
  - - partyChoice : CPLParty
- Methods:
  - + CPLBallot(partyChoice : CPLParty)

    - + getPartyChoice() : CPLParty

CPLElection

- Attributes:
    - - parties : CPLParty[]
    - - numSeats : int
    - - results : String[]
- Methods:
    - + CPLElection(parties : CPLParty[], ballots: Ballot[], numballots: int, numSeats: int)
    - + getNumParties() : int
    - + getNumSeats() : int
    - + assignSeats() : void
    - + runElection() : void
    - + printElectionResults() : void
    - + produceAuditFile() : void
    - + getResults() : String[]

CPLFileProcessor

- Methods:
    - + CPLFileProcessor()
    - + processFile(inputFile : File) : CPLElection

CPLParty

- Attributes:
    - - rankedPartyCandidates : String[]
    - - numPartyCandidates : int
    - - numSeatsAllotedFirst : int
    - - numSeatsAllotedSecond : int
    - - numVotesAfterFirstAllocation : int
- Methods:
    - + CPLParty(name : String, ID : int, candidates : String[])
    - + getPartyCandidates() : String[]
    - + getNumPartyCandidates() : int
    - + getTopPartyCandidates(numSeats : int) : String[]
    - + getNumSeatsAllotedFirst() : int
    - + setNumSeatsAllotedFirst(seats : int) : void
    - + getNumSeatsAllotedSecond() : int
    - + setNumSeatsAllotedSecond(seats : int) : void
    - + getNumVotesAfterFirstAllocation() : int
    - + setNumVotesAfterFirstAllocation(num : int) : void

<<abstract>> Election

- Attributes:
  - # votables: Votable[]
  - # numVotables: int
  - # ballots: Ballot[]
  - # numBallots : int
- Methods:
  - + *runElection() : void*
  - + *printElectionResults() : void*
  - + getVotables: Votable[]
  - + getNumVotables: int
  - + getNumBallots: int
  - + *produceAuditFile(): void*
  - + breakTie(votable : Votable[]) : Votable[]
  - + promptUserForDate() : String

File: File(Java Platform SE 7) (oracle.com)

FileHandler

- Attributes:
  - - filename : String
- Methods:
  - + FileHandler()
  - + FileHandler(filename : String)
  - + getFileNameFromInput() : void
  - + openFile() : File

<<abstract>> FileProcessor

- Attributes:
  - # ballots : Ballot[]
- Methods:
  - + processFile(inputFile: File) : Election

IRBallot

- Attributes:
  - - candidatesQueue : PriorityQueue<IRCandidate>
- Methods:
  - + IRBallot(rankedCandidates : IRCandidates[])
  - +redistributeVote(): void

IRCandidate

- Attributes:
    - - voteCountHistory : ArrayList<integer>
- Methods:
    - + IRCandidate(name : String, ID : int)
    - + updateVoteCountHistory() : void
    - + getVoteCountHistory(): ArrayList<integer>

IRElection

- Attributes:
    - - rankedCandidates : IRCandidate[]
    - - eliminatedCandidates: IRCandidate List
- Methods:
    - + IRElection(rankedCandidates : IRCandidate[], ballots : Ballot[])
    - + getNumCandidates() : int
    - + getNumBallots() : int
    - + getCandidates() : IRCandidate[]
    - + eliminateCandidate(cand : IRCandidate) : void
    - + reRankCandidates() : void
    - + runElection() : void
    - + redistributeVotes(cand: IRCandidate) : void
    - + printElectionResults() : void
    - + produceAuditFile() : void

IRFileProcessor

- Methods:
    - + IRFileProcessor()
    - + processFile(inputFile : File) : IRElection

Main

- Methods:
    - + main(String[] args) : void

<<Abstract>> Voteable

- Attributes:
    - # name: String
    - # numVotes: int
    - # ID : int
- Methods:
    - + getName() : String
    - + getNumVotes() : int
    - + getID() : int

    - + incrementVotes(amount : int) : void

5. Component Design

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

<<Abstract>>Ballot

CPLBallot:

- Attributes:
    - - partyChoice : CPLParty

Methods:

```
+ CPLBallot(partyChoice : CPLParty)
        this.partyChoice = partyChoice
+ getPartyChoice() : CPLParty
        Return partyChoice
```

CPLElection

Attributes:

- parties : CPLParty[]

- numSeats : int

- results : String[]

Methods:

```
+ CPLElection(parties : CPLParty[], ballots: Ballot[], numballots: int, numSeats: int)
        this.parties = parties
        this.ballots = ballots
        this.numBallots = numballots
        this.numSeats = numSeats
+ getNumParties() : int
        Return length(this.parties)
```

```
+ getNumSeats() : int
        Return this.numSeats
+ getResults() : String[]
        Return this.results
+ assignSeats() : void
        quota = numBallots / numSeats
        Seatsallocated = 0
        For party in parties:
                seats = floor(party.getNumVotes() / quota)
                Seatsallocated += seats
                party.setNumSeatsAllotedFirst(seats)
                party.setNumVotesAfterFirstAllocation(party.getNumVotes() - quota*seats)
        // second allocation of seats
        While seatsallocated < numSeats:
                Max = -1
                Maxparty = null
                maxTie = CPLParty[numParties] // tracks all parties tied for top
                For party in parties:
                        If party.getNumVotesAfterAllocation() > max
                                Max = party.getNumVotesAfterAllocation()
                                Maxparty = party
                                maxTie.clear()
                                maxTie[0] = maxparty
                        Else if party.getNumVotesAfterAllocation() == max
                                i = 1
                                While i < numParties and maxTie[i] != Null
                                        i ++
                                maxTie[i] = party
                Maxparty = breakTie(maxTie) //breaks tie if necessary, gets winner
                Seatsallocated += 1
                Maxparty.setNumSeatsAllotedSecond(1)
                Maxparty.setNumVotesAfterFirstAllocation(-1)
        // all seats now allocated. Now add to results[]
        k = 0
        For party in parties:
                For i in range(party.getNumSeatsAlloted()):
                        results[k] = party.getPartyCandidates()[i]
                        k += 1
+ runElection() : void
        winner = null
        For ballot in ballots:
                party = ballot.getPartyChoice()
                party.incrementVotes(1)
        assignSeats()


+ printElectionResults() : void
        print("the winners are: ")
        For candidate in results:
                print(candidate)
+ produceAuditFile() : void
        fp = fileopen("auditfile.txt", "w")
        fp.write(("-" * 50) + "\n")
        fp.write("Parties \t Votes \t First Allocation of Seats \t Remaining Votes \t Second allocation
of seats \t final seat total \t % of vote to % of seats \n")
        fp.write(("-" * 50) + "\n")
        for party in parties:
                fp.write(party.getName() + "\t")
                fp.write(party.getNumVotes() + "\t")
                fp.write(party.getNumSeatsAllotedFirst() + “\t”)
                fp.write(party.getNumVotesAfterFirstAllocation() + “\t”)
                fp.write(party.getNumSeatsAllotedSecond() + “\t”)
                partyTotalSeats = party.getNumSeatsAllotedFirst() +
        party.getNumSeatsAllotedSecond()
                fp.write(partyTotalSeats + “\t”)
                percSeats = totalSeats / numSeats
                percVote = party.getNumVotes() / numBallots
                fp.write(percVote + “% / ” + percSeats + “% \n”
        fp.write((“-” * 50) + “\n”)
        fp.close()
```

CPLFileProcessor

Methods:

+ CPLFileProcessor()

+ processFile(inputFile : File) : CPLElection

```
    s = new Scanner(inputFile)
    s.nextLine() // skip over line specifying election type
    numParties = s.nextLine()
    partiesStrings = s.nextLine().split(“,”)
    parties = CPLParty[numParties]
    for i in range(numParties):
        cands = s.nextLine().split(“,”)
        parties[i] = new CPLParty(partiesStrings[i], i, cands)
    numSeats = s.nextLine()
    numBallots = s.nextLines()
    ballots = CPLBallot[numBallots]
    for i in range(numBallots):
        arr = s.nextLine().split(“,”)
        k = -1
        for j in range(arr.length()):
            if arr[j] != “”:
                k = j
        ballots[i] = new CPLBallot(parties[k])
    s.close()
    election = new CPLElection(parties, ballots, numBallots, numSeats)
    return election
```

CPLParty

Attributes:

- rankedPartyCandidates : String[]
- numPartyCandidates : int
- numSeatsAllotedFirst : int
- numSeatsAllotedSecond : int
- numVotesAfterFirstAllocation : int

Methods:

```
+ CPLParty(name : String, ID : int, candidates : String[])
    this.name = name
    this.ID = ID
    this.rankedPartyCandidates = candidates
    this.numVotes = 0
    this.numPartyCandidates = candidates.length()
    this.numSeatsAlloted = 0
+ getPartyCandidates() : String[]
    return this.rankedPartyCandidates
+ getNumPartyCandidates() : int
    return this.numPartyCandidates
+ getTopPartyCandidates(numSeats : int) : String[]
    return this.numPartyCandidates[0:numSeats]
+ getNumSeatsAllotedFirst() : int
```

return this.numSeatsAllotedFirst

+ setNumSeatsAllotedFirst(seats : int) : void

this.numSeatsAllotedFirst = seats

+ getNumSeatsAllotedSecond() : int

return this.numSeatsAllotedSecond

+ setNumSeatsAllotedSecond(seats : int) : void

this.numSeatsAllotedSecond = seats

+ getNumVotesAfterFirstAllocation() : int

return this.numVotesAfterFirstAllocation

+ setNumVotesAfterFirstAllocation(num : int) : void

this.numVotesAfterFirstAllocation = num

<<Abstract>>Election

Attributes:

# votables: Votable[]

# numVotables: int

# ballots: Ballot[]

# numBallots : int

Methods:

+ runElection() : void

Return

+ printElectionResults() : void

Return

+ getVotables: Votable[]

Return this.votables

+ getNumVotables: int

Return this.numVotables

+ getNumBallots: int

Return this.numBallots

+ produceAuditFile(): void

```
    *Return*
+ breakTie(votable : Votable[]) : Votable
    rand = 0.0
    for i in range(1000)
        rand = random(0 to 1)
    rand_votables = int[votable.length]
    For i in range(votable.length):
        if votable[i] != Null
            rand_votables[i] = random(0 to 1)
        else
            Rand_votables[i] = 3 //large enough number that it won't interfere with 0 to 1
    minDist = 2
    minIdx = -1
    for i in range(votable.length)
        dist = abs(rand - rand_votables[i])
        if dist < minDist
            minDist = dist
            minIdx = i
    return votable[minIdx]
+ promptUserForDate() : String
    String userInput = input("Enter the date of the election: ")
    Return userInput
```

File: File(Java Platform SE 7) (oracle.com)

FileHandler

- Attributes:
  - - filename : String

Methods:

```
+ FileHandler()
    This.filename = Null
```

```
+ FileHandler(filename : String)
    This.filename = filename
+ getFileNameFromInput() : void
    this.filename = input("Enter the name of the CSV file containing ballot information: ")


+ openFile() : File
    If this.filename == Null:
        getFileNameFromInput()
    File fp = fileopen(this.filename)
    Return fp
```

<<Abstract>>FileProcessor

- Attributes:
    - # ballots : Ballot[]

Methods:

```
+ processFile(inputFile: File) : Election
    return
```

IRBallot

- Attributes:
    - - candidatesQueue : PriorityQueue<IRCandidate>

Methods:

```
+ IRBallot(rankedCandidates : IRCandidates[])
    this.candidatesQueue = rankedCandidates
+ redistributeVote() : void
    // Pops candidate from the front of the queue, making the new number 1 vote the second in
    // the queue. Also if
    this.candidatesQueue.pop()


+ getFirstCandidate()
    return
```

IRCandidate

- Attributes:
  - -voteCountHistory : ArrayList<Integer>

Methods:

+ IRCandidate(name : String, ID : int)

```
    This.name = name
    This.ID = ID
```

+ updateVoteCountHistory() : void

```
    //Updates Vote count history
    voteCountHistory.append(this.getNumVotes())
```

+ getVoteCountHistory() : ArrayList<Integer>

```
    return this.voteCountHistory
```

IRCandidateQueue

- Methods:

IRElection

- Attributes:
  - - rankedCandidates : IRCandidate[]
  - - eliminatedCandidates : IRCandidate List

Methods:

+ IRElection(rankedCandidates : IRCandidate[], ballots : Ballot[], eliminatedCandidates : IRCandidate[] = [])

```
    this.rankedCandidates = rankedCandidates
    This.ballots = ballots
```

+ getNumCandidates() : int

```
    Return numVotables
```

+ getNumBallots() : int

```
    Return numBallots
```

+ getCandidates() : IRCandidate[]

```
    Return this.rankedCandidates
```

```
+ eliminateCandidate(cand : IRCandidate) : void
        this.rankedCandidates.remove(cand)
        this.votables.remove(cand)
        this.numVotables -= 1;
+ reRankCandidates() : void
        Sort by numVotes(this.rankedCandidates)
        //Check for ties, use breakTie() to break any ties and get the order
+ runElection() : void
        // Initial input for BallotHistory - for audit file
        for cand in rankedCandidates:
                cand.updateVoteCountHistory()
        Int total = getNumBallots()
        While this.rankedCandidates[0].getNumVotes < total//2 && length(elminatedCandidates) <
                                                                        numVotables - 2:
                If this.rankedCandidates[-1].getNumVotes() ==
                                                this.rankedCandidates[0].getNumVotes():
                        breakTie(this.rankedCandidates)
                        Break
                Else:
                        RedistributeVotes()
                reRankCandidates()
+ RedistributeVotes(): void
        // Redistribute votes of candidate in last
        cand = this.rankedCandidates[-1]
        eliminatedCandidates.append(cand)
        for (int i = 0; i < length(ballots); i++):
                If ballots[i].getFirstCandidate.ID() == cand.ID:
                        ballots[i].redistributeVote()
                        While (ballots[i].getFirstCandidate() in eliminatedCandidates):
```

```
                ballots[i].redistributeVote()
    for cand in rankedCandidates:
        cand.updateVoteCountHistory()
+ printElectionResults() : void
    voteCount = this.rankedCandidates[0].getNumVotes()
    print(this.rankedCandidates[0], " wins with ", voteCount, " votes \n")
    print("Check generated audit file for a look at the specific steps of an election \n")

+ produceAuditFile() : void
    fp = fileopen("auditfile.txt", "w")
    // Add in initial round, where nobody is eliminated
    fp.write("Round 0:\n")
        for candidate in rankedCandidates()
            Ct = candidate.getVoteCountHistory().get(0)
            fp.write(candidate.getName() + ": " candidate.getVoteCountHistory().get(0))
            + "\n")
    for i in range(length(eliminatedCandidates))
        fp.write("Round" + (i + 1) + ": " +  eliminatedCandidates.get(i) + " eliminated\n")
        for candidate in rankedCandidates()
            Ct = candidate.getVoteCountHistory().get(i + 1)
            If Ct == 0:
                fp.write("-------")
            else:
                fp.write(candidate.getName() + ": "
            candidate.getVoteCountHistory().get(i + 1)) + "\n")
    fp.write("Winning Candidate: " + rankedCandidates[0].getName())
    fp.close()
```

IRFileProcessor

Methods:

```
+ IRFileProcessor()
+ processFile(inputFile : File) : IRElection
    s = new Scanner(inputFile)
    s.nextLine() // skips line containing text IR
    numCand = int(s.nextLine())

    candString = s.nextLine().split(“,”)
    Candidates = IRCandidate[numCand]
    for i in range(numCand):
        Candidates[i] = new IRCandidate(candString[i], i)

    numBallots = s.nextLine()
    ballots = IRBallot[numBallots]
    for i in range(numBallots):
        arr = s.nextLine().split(“,”)
        ballots[i] = new IRBallot(arr)
    s.close()
    Candidates = sort by numVotes(Candidates)
    election = new IRElection(Candidates, ballots)
    return election
```

Main

Methods:

```
+ main(String[] args) : void
    FileHandler fh
    If args == Null:
        Fh = new FileHandler()
    Else:
        Fh = new FileHandler(args[0])
    File fp = fh.openFile()
    FileProcessor fileP = new FileProcessor()
    Election currentElection = fileP.processFile(fp)
    currentElection.runElection()
    currentElection.printElectionResults()
    currentElection.produceAuditFile()
```

<<Abstract>> Voteable

- Attributes:
  - - name: String
  - - numVotes: int
  - - ID : int

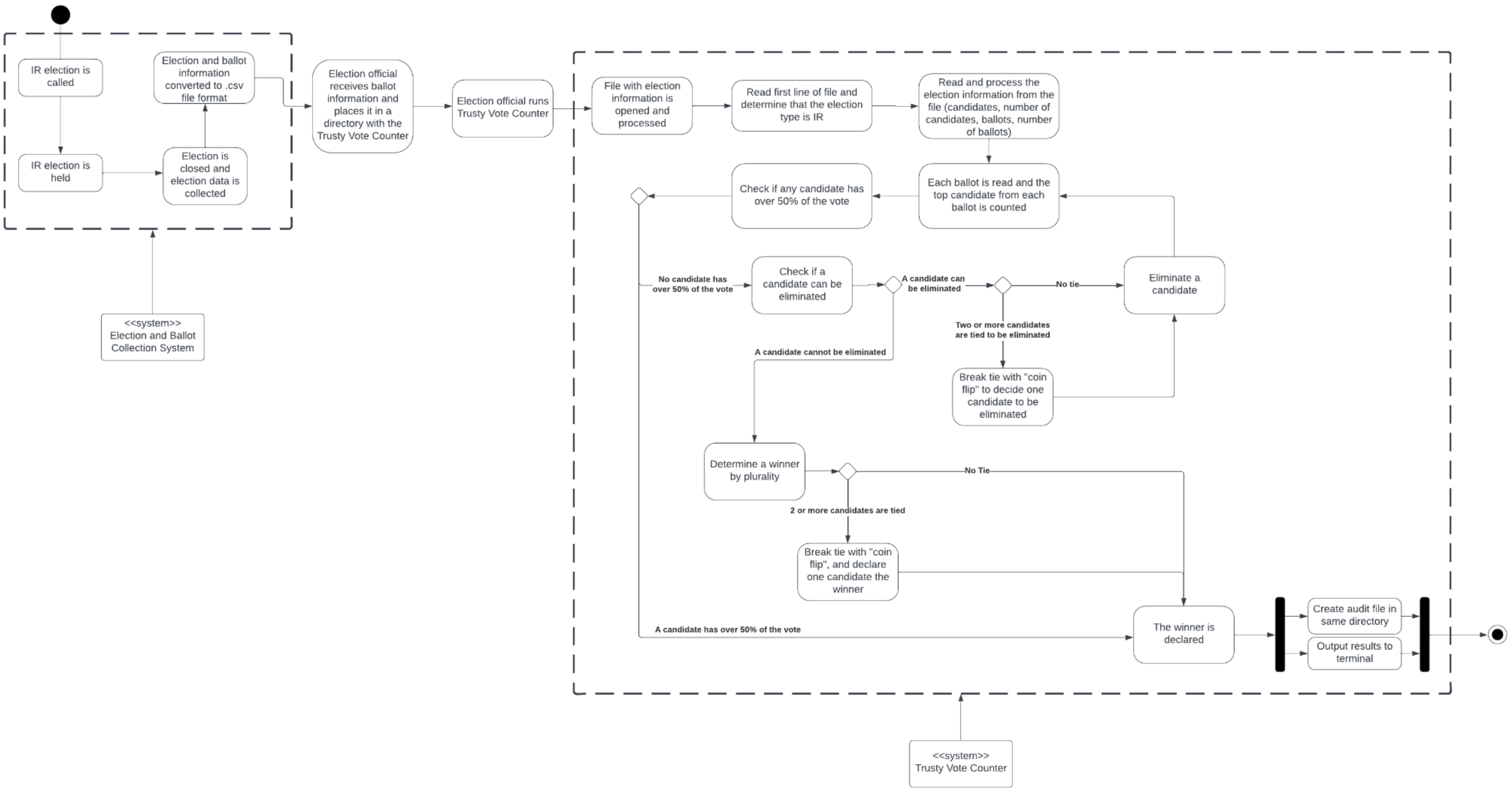
Methods:

```
getName() : String
    Return this.name
+ getNumVotes() : int
    Return this.numVotes
+ getID() : int
    Return this.ID
+ incrementVotes(amount : int) : void
    this.numVotes += amount
```

6. Human Interface Design

6.1 Overview of User Interface

To use this program, the program is run in a terminal. A user (pictured as an election official in the activity diagram below) runs the program on a terminal, and the results are displayed in the terminal. An audit file is created in the same directory. An example of these can be seen in section 6.2.

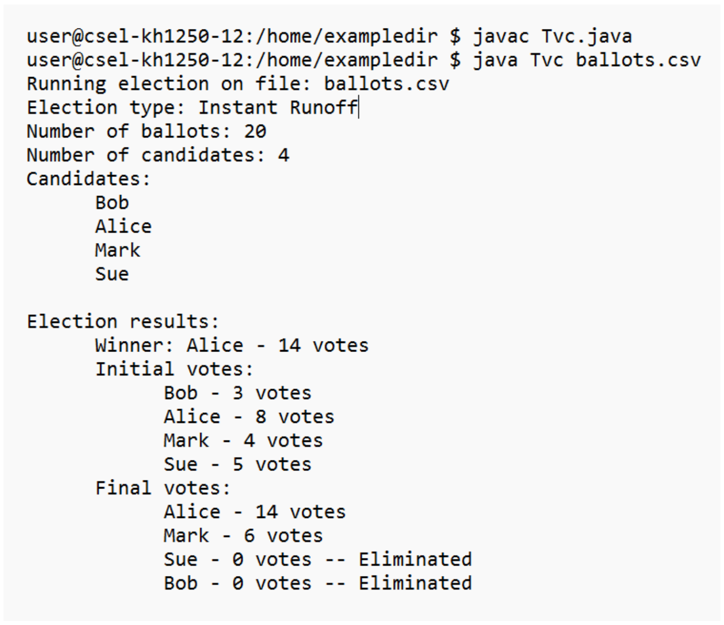

IR election is called
IR election is held
Election is closed and election data is collected
Election and ballot information converted to .csv file format
<<system>> Election and Ballot Collection System
Election official receives ballot information and places it in a directory with the Trusty Vote Counter
Election official runs Trusty Vote Counter
File with election information is opened and processed
Read first line of file and determine that the election type is IR
Read and process the election information from the file (candidates, number of candidates, ballots, number of ballots)
Each ballot is read and the top candidate from each ballot is counted
Check if any candidate has over 50% of the vote
No candidate has over 50% of the vote
Check if a candidate can be eliminated
A candidate can be eliminated
No tie
Eliminate a candidate
Two or more candidates are tied to be eliminated
Break tie with "coin flip" to decide one candidate to be eliminated
A candidate cannot be eliminated
Determine a winner by plurality
No Tie
2 or more candidates are tied
Break tie with "coin flip", and declare one candidate the winner
A candidate has over 50% of the vote
The winner is declared
Create audit file in same directory
Output results to terminal
<<system>> Trusty Vote Counter

Note: For a bigger image, see the github repo linked in the appendices section. Navigate to the SDS folder and open SDDActivityModel.png

6.2 Screen Images

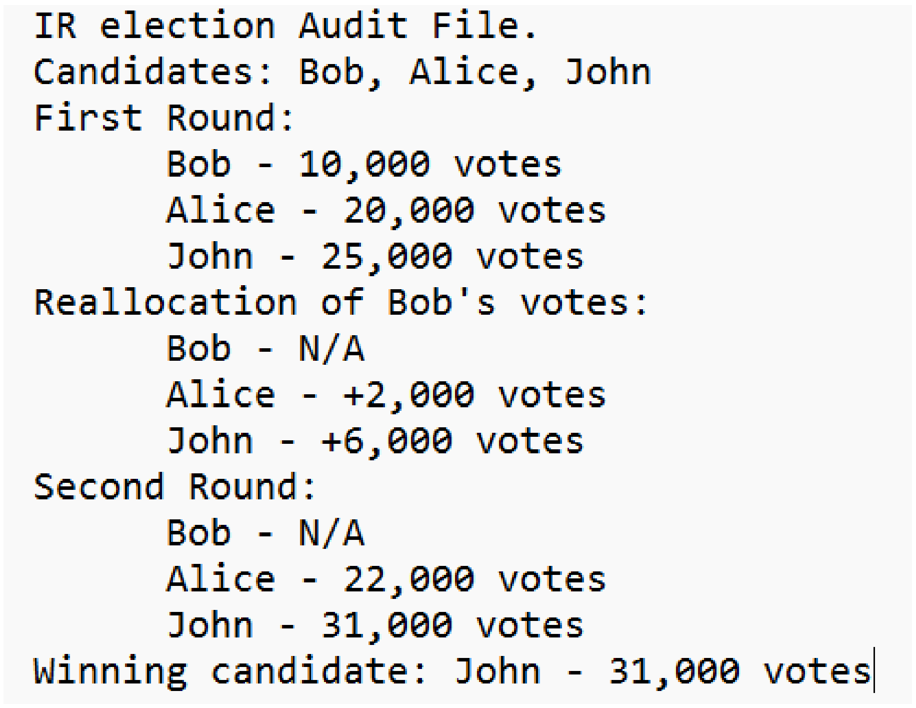
Below is an example of running the *Trusty Vote Counter* program. For more details see section 6.3

Example audit file output for the IR election



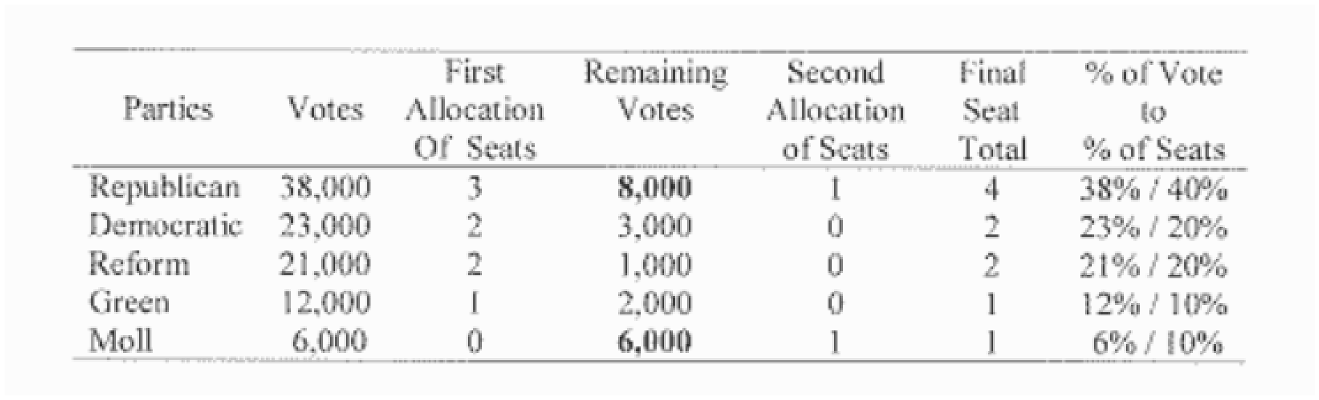
```
user@csel-kh1250-12:/home/exampledir $ javac Tvc.java
user@csel-kh1250-12:/home/exampledir $ java Tvc ballots.csv
Running election on file: ballots.csv
Election type: Instant Runoff
Number of ballots: 20
Number of candidates: 4
Candidates:
      Bob
      Alice
      Mark
      Sue

Election results:
      Winner: Alice - 14 votes
      Initial votes:
            Bob - 3 votes
            Alice - 8 votes
            Mark - 4 votes
            Sue - 5 votes
      Final votes:
            Alice - 14 votes
            Mark - 6 votes
            Sue - 0 votes -- Eliminated
            Bob - 0 votes -- Eliminated
```

Example Audit file for IR election:



```
IR election Audit File.
Candidates: Bob, Alice, John
First Round:
      Bob - 10,000 votes
      Alice - 20,000 votes
      John - 25,000 votes
Reallocation of Bob's votes:
      Bob - N/A
      Alice - +2,000 votes
      John - +6,000 votes
Second Round:
      Bob - N/A
      Alice - 22,000 votes
      John - 31,000 votes
Winning candidate: John - 31,000 votes
```

Example audit file output for CPL Election:



Parties	Votes	First Allocation Of Seats	Remaining Votes	Second Allocation of Seats	Final Seat Total	% of Vote to % of Seats
Republican	38,000	3	**8,000**	1	4	38% / 40%
Democratic	23,000	2	3,000	0	2	23% / 20%
Reform	21,000	2	1,000	0	2	21% / 20%
Green	12,000	1	2,000	0	1	12% / 10%
Moll	6,000	0	**6,000**	1	1	6% / 10%

6.3 Screen Objects and Actions

The most important object on the screen is the terminal which will be the main way that the user will be interacting with the program. The terminal should be auto generated in the IDE for most environments, and the user should be able to use their keyboard and mouse to type in the terminal. As you can see from the image above in section 6.2, the first few lines are examples of how the Trusty Vote Counter (TVC) is run, which is by a text input along with a file name. When the user is prompted to enter information like a file name, the user will see a flashing text cursor within the terminal, at which the user should enter the information that is being prompted.

After the part in which the user manually inserts the relevant information into the terminal, the program will run automatically and output text information like the election results and candidates shown in the image above. There is no need for the user to perform any action at this point, and the program will automatically exit when it is finished outputting the results, or might prompt the user to confirm the exit with a key press.

7. Requirements Matrix

Use Case (From SRS)	System Component
UC_001	File, FileHandler
UC_002	FileProcessor
UC_003	IRFileProcessor
UC_004	CPLFileProcessor
UC_005	IRElection, IRCandidateQueue, Main
UC_006	IRElection, IRBallot

UC_007	IRElection
UC_008	CPLElection, CPLBallot
UC_009	CPLElection, CPLBallot, CPLParty
UC_010	IRElection, CPLElection
UC_011	IRElection, CPLElection
UC_012	Main, IRElection, CPLElection
UC_013	Main, File
UC_014	Main, IRElection, CPLElection
UC_015	Main, File
UC_016	Main, FileHandler, FileProcessor, File

8. Appendices

The Software Requirements Specification by Robert Wang, Alex Iliarski, Ethan Loukusa, Gideon Tan
https://docs.google.com/document/d/1W9HJ30_GzC4F3dRuatKkImuk7_hfZnDh/edit?usp=sharing&ouid=117208831313787612273&rtpof=true&sd=true

TVC Github Repo - Includes SRS, SDS, code, and supporting images and documents.

https://github.umn.edu/umn-csci-5801-02-S23/repo-Team16

Lucid Chart for the UML class diagram

https://lucid.app/lucidchart/086811bf-f5c0-4ad2-bba4-29bd5e462b94/edit?viewport_loc=-238%2C1320%2C1732%2C823%2C0_0&invitationId=inv_8075e944-4003-45fd-91ad-d93f0f1cfc1c

Lucid Chart for the CPL sequence diagram

https://lucid.app/lucidchart/c0e2f662-299a-4374-ab37-cc31029e23bc/edit?viewport_loc=826%2C-154%2C3464%2C1646%2C0_0&invitationId=inv_6144faf4-2436-4384-a8ed-e8494b56c2cb

Lucid Chart for the IR Activity Model

https://lucid.app/lucidchart/aa45944a-66c7-4527-a7d6-c2ca16b4f219/edit?viewport_loc=172%2C120%2C3464%2C1646%2C0_0&invitationId=inv_1ac679bf-4df3-4677-aecc-63d5c4a933ce