Team 6

Voting System

Software Design Document

Names: Eileen Campbell, Maranda Donaldson, Hazel Dunn, Olivia Hansen
x500: camp0870, donal163, dunn0441, hans6055

Date: February 28, 202

1. INTRODUCTION

1.1. Purpose

This document contains the design specifications for our Voting System. The expected audience is the programmers working on the implementation of the system. It could also be useful for anyone who wants to use the system or make changes in the future.

1.2. Scope

This document contains a complete description for the design of a voting system. This system can analyze data for both an Instant Runoff or Open Party List election. It will take in data for a specific election and export election results to a media file, an audit file, and the terminal. The program is written in Java and all user interaction will be through a computer terminal. This program drastically reduces human error and the amount of time it takes to calculate the results of an election. There are no security precautions. Any user with a copy of the program has the authority to obtain election results.

1.3. Overview

This document will be organized into eight sections. The first section will contain an overview of the whole system: its purpose, the goals of the program, what the system does, references to any cited outside material, and definitions of terms that may be unfamiliar to the reader.

The second section is an overview of the whole system and how it operates. Compared to the previous section, it will provide more detail about the software itself, such as descriptions of functionality, context, and design of the program.

Section three will provide detail about the architecture of the system, how tasks are split into subsystems, and the reasoning behind the design.

The fourth section will describe the data, and how it is transformed into data structures. It will lay out how the objects are stored, processed, and organized.

Section number five will focus on the component design. It will go into depth about what each component does in the system. A description of class functions and objects will be provided.

The sixth section will focus less on the details and operations of the systems themselves, and will instead focus on the user interaction with the system through displays and feedback. The section also provides example screenshots of the program.

Section seven will be a simple table that goes through each requirement that the user and designers specified for the program in the SRS document, and traces components and data structures of the system to them.

Section number eight will be appendices that provide any additional information that could aid the user in understanding this document: the class diagram, Instant Runoff election activity diagram, Open Party List election activity diagram, and Open Party List election sequence diagram.

1.4. Reference Material

Instant Runoff Voting Information:
https://www.fairvote.org/glossary

Part List Voting Information:
https://www.fairvote.org/how_proportional_representation_elections_work

Software Designment Document Template (provided by Dr. Shana Watters):
https://drive.google.com/file/d/1b8HCb5cIb10WkQvJl4IVux19gK_NbZjx/view?usp=sharing

1.5. Definitions and Acronyms

- Instant Runoff Voting (IR): All candidates are listed on the ballot and voters rank candidates in order of their preference.
- Open Party List Voting (OPL): Voters are presented an unordered list of candidates chosen in party primaries. Voters cast a vote for individual candidates. This vote counts for the specific candidate as well as for the party. Number of seats are delegated by which party got the most votes and which candidates were the most popular in each party.
- CSV File: Comma Separated Values. A plain text file that contains a list of data. Every value is separated by a comma.

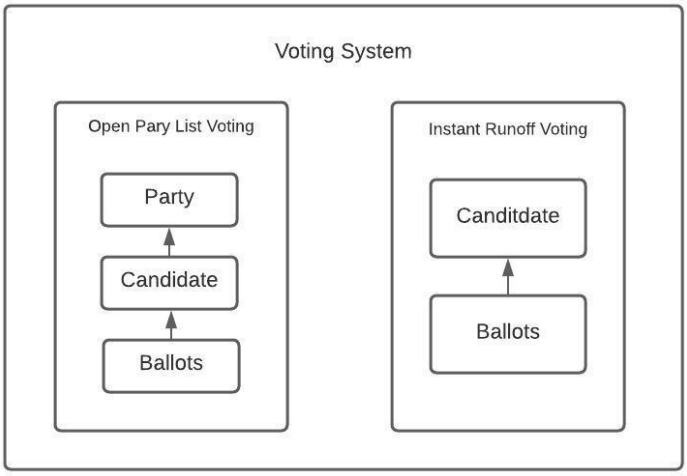
2. SYSTEM OVERVIEW

This system's primary purpose is to analyze ballot data to determine an election's results. In order to accomplish this, the system reads in data from a CSV file, stores this data in an efficient manner, runs one of two specified algorithms to determine the election results, and prints the results to multiple files and other outputs. To read in data from a given CSV file, the system prompts the user to input the file name. While getting user input, the system also prompts the user to see what to call the two output files. After getting all user inputs, the system then reads in all necessary data from the CSV file, using the first line of the file to determine if it will run the IR or OPL voting algorithm. While running the algorithms, the system continually prints to the audit output file to show the progress of the election. Once the algorithms have completed, a summary of results will be printed to the screen and to a media report.

3. SYSTEM ARCHITECTURE

3.1. Architectural Design

The software is designed to handle two different types of voting algorithms. The first step in the decomposition process is to split the voting system into two subsystems: OPL and IR. From here, we decomposed each subsystem. The OPL algorithm first allocates seats based on party, and then by candidate. Because of this process, parties will be created first in the subsystem followed by instantiation of candidate objects. Candidates have ballots assigned to them. In IR, only candidate and ballot objects are needed. Candidates also have ballots assigned to them, similar to OPL. The breakdown of the subsystems is depicted in the diagram below.



3.2. Decomposition Description

The system is an object oriented design. Appendix A, the Class Diagram, can be referenced to track the types of objects. The document, UseCase_Team6 (linked in Appendix E), can be referenced for a detailed description of each case. The program begins as a Voting System. This prompts the user for the CSV file name (UC_001), the audit file name (UC__003), and the media file name (UC_004). The first line of the CSV file will be read (UC_002) which will determine whether to create an OPLElection object or an IRElection object (UC_005). These objects are responsible for the algorithm for the specified election. At any point if a tie occurs in either election, a coin toss will be used to determine how to proceed. If the tie is for the lowest votes, the selected object is eliminated. If the tie is for the highest number of votes, the selected object is the winner (UC_007). Each election object is responsible for writing to an audit file (UC_009), media file (UC_010), and printing the results to the user when the algorithm completes (UC_011).

The IRElection creates candidate and ballot objects. A loop will be used to check if a candidate has received a majority of votes. If no majority is found, it is responsible for redistributing the losing candidate's ballots to the next ranked choice. If no candidate receives a majority number of votes, the most popular candidate is declared the winner (UC_008).

The OPLElection creates party, candidate, and ballot objects. Party objects contain a list of candidates in order of popularity. Any independent candidate will be grouped into an “Independent” party (UC_006). Once the seats have been allocated, it will be easy to access the party objects and assess which candidates won the party seats.

3.3. Design Rationale

First of all, we chose to have an overarching “voting system” class that covers the basics of both types of elections without going into specifics, so it does the general work. Then we have this general class split off into our two specific voting systems, OPL and IR, which each operate independently of each other. We chose this architecture because it allows us to branch off of a main system and operate our unique voting systems when

appropriate. In the future, this layout will allow new voting systems to be added if desired. We can keep the general system as is and it will work for other voting systems, which would just need to be added as separate classes like our IR or OPL.

We then have our general party, candidate, and ballot classes, which we chose to do so they can be used by both systems that we have, and any system we choose to add in the future. We then created small separate classes for each type of ballot, which could easily be done to create a new one if an additional system was added. Having the general overarching voting system as well as general party, candidate and ballot classes allows both of our elections to use the same structures, as well as any that wish to be added down the road, with few modifications necessary.

We chose to use the command line to prompt the user instead of a GUI because of the simplicity of the access to the user, and the ease of programming. It allows direct communication to the user through a command line which is widely available on most devices.

4. Data Design

4.1. Data Description

All information used by the voting system will be read in from a CSV file. Broad data, such as total number of votes and number of candidates, will be stored as variables in the voting system class. When reading in the list of candidates, Candidate objects will be created and contain information pertaining to the individual candidate. These objects will hold and group ballot objects. If an OPL file is read, a similar process occurs with the party information. When a ballot line is read, a ballot object is created of the correct type (OPL or IR) and all information will be stored in that object. To connect ballots to candidates, each candidate object will have an ArrayList of ballots. For the OPL algorithm, party objects will have an ArrayList of candidates that belong to that party. The full list of candidates will be stored as an ArrayList inside the voting system. These objects will be accessed at the end of a run of the system to write pertinent information to the media and audit files, as well as print a summary of the election to the screen.

4.2. Data Dictionary

Voting System

Attribute Name	Attribute/Return Type	Description
auditFile	PrintWriter	Used to write to the audit file
candidates	ArrayList<Candidate>	Used to keep track of the candidates in the CSV file in the order they appear
coinToss(int numTied)	int	Used to break ties throughout both voting systems
csvFile	Scanner	Used to read from the CSV input file

csvName	String	Used to store the CSV input file name
electionType	String	Used to store the election type from the first line of the CSV file
main()	void	Used to run the whole system
mediaFile	PrintWriter	Used to write to the media file
promptAudit()	void	Used to read in the audit file name from the user or generate a unique name and open/create the file for writing
promptCSV()	void	Used to read in the CSV file name from the user and open the file for reading
promptMedia()	void	Used to read in the media file name from the user or generate a unique name and open/create the file for writing
totalNumBallots	int	Used to store the total number of ballots in the CSV file
startTime	long	Used to track the system time of the program
stopTime	long	Used to track the system time of the program

OPL Election

Attribute Name	Attribute/Return Type	Description
allocateByQuota()	void	Used to do the first round seat allocation to parties
allocateByRemainder()	void	Used to allocate the remaining seats to parties after the first round
numSeatsLeft	int	Used to keep track of the number of seats left to allocate
party	ArrayList<String>	Used to keep track of all the parties in the election

partyNumBallots()	void	Used to calculate the number of ballots for each party in the party ArrayList
printToScreen()	void	Used to print the election results to the screen
quota	int	Used to store the vote quota for the election
readBallots()	void	Used to read in the ballots from the CSV file
readOPLCSV()	void	Used to read in the election information for the CSV file
runElection()	void	Used to start running the OPL algorithm
totalNumSeats	int	Used to store the total number of seats to allocate for the election
writeToAuditFile()	void	Used to write the necessary information to the audit file
writeToMediaFile()	void	Used to write the necessary information to the media file

IR Election

Attribute Name	Attribute/Return type	Description
currCandidates	ArrayList <Candidate>	Used to keep track of the candidates who are still in the race
eliminatedCandidates	ArrayList<Candidate>	Used to keep track of the candidates who have been eliminated
findLeastCan()	Candidate	Used to find the candidate with the least amount of votes
findMajority()	Candidate	Used to find the candidate with the majority of the votes
printScreen()	void	Used to print the election results to the screen

readBallots()	void	Used to read in the ballots from the CSV file
readIRCSV()	void	Used to read in the election information from the CSV file
redistributeBallots()	void	Used to redistribute the ballots of the losing candidate to their next choice
runElection()	void	Used to start running the IR algorithm
writeToAuditFile()	void	Used to write the necessary final information to the audit file
writeToAuditFile(Candidate c)	void	Used to write the necessary information about an eliminated candidate to the audit file
writeToMediaFile()	void	Used to write the necessary information to the media file

Party

Attribute Name	Attribute/Return Type	Description
addCandidate(Candidate)	void	Used to add a candidate object to the ArrayList attribute, candidates, in the Party object
calculateNumBallots()	void	Used to calculate the total number of ballots for the party by summing up each candidates ballots
candidates	ArrayList<Candidate>	Used to keep track of the candidates in the party
coinToss(int numTied)	int	Used to break ties throughout both voting systems
numSeats	int	Used to store the number of seats allocated to a party
pName	string	Used to store the name of the party
pNumBallots	int	Used to store the total number of ballots for the party

remainder	int	Used to store the remainder for the party
sortCandidates()	void	Used to sort the candidates ArrayList by number of ballots

Candidate

Attribute Name	Attribute/Return Type	Description
addBallot(Ballot)	void	Used to add a ballot to the candidate's ballot ArrayList
cBallots	ArrayList<Ballot>	Used to store all of the ballots allocated to a candidate
cName	string	Used to store a candidate's name
cNumBallots	int	Used to store the total number of ballots allocated to a candidate
cParty	string	Used to store the party the candidate is running for

Ballot

Attribute Name	Attribute/Return Type	Description
privateID	int	Used to store the unique ID to each ballot

OPL Ballot

Attribute Name	Attribute/Return Type	Description
bCandidate	Candidate	Used to store the Candidate that a ballot voted for
bParty	string	Used to store the part of the candidate that a ballot voted for

IR Ballot

Attribute Name	Attribute/Return Type	Description
ranking	ArrayList<Candidate>	Used to store the candidate ranking of a ballot
rankIndex	int	Used to store the index of the earliest candidate in the ranking array that is still in the race
numCandidates	int	Used to keep track of the number of candidates in the ranking ArrayList since size is not always reliable

5. Component Design

Below is a description of each function in each subsystem. Detailed diagrams can be referenced in appendix A: Class Diagram, Appendix B: Instant Runoff Activity Diagram, Appendix C: Open Party List Election Activity Diagram, and Appendix D: Open Party List Sequence diagram.

5.1. Voting System

- promptAudit()
  This function will prompt the user for the desired audit file name. If one is not provided, the function will generate a unique name. It will then create the file and open it for writing. It then sets the class variable auditFile to the PrintScanner to be accessed and written to throughout the algorithm.
- promptMedia()
  This function will prompt the user for the desired media file name. If one is not provided, the function will generate a unique name. It will then create the file and open it for writing. It then sets the class variable mediaFile to the PrintScanner to be accessed and written to throughout the algorithm.
- promptCSV()
  This function will prompt the user for the name of the CSV containing the election information. It will then open the file and set the class variable csvName to the string input by the user, and the class variable csvFile to the Scanner to be read from throughout the program. Lastly, it will read in the first line of the file and set the class variable electionType.
- CoinToss(int num) : int
  This function will be used to break any ties throughout either algorithm. It takes in the number of candidates involved in the tie, and returns an integer representing the losing candidate. It will simulate 1000 coin tosses and take the 1001 result to ensure randomness.
- main()

This function will be used to call all of the other functions and run the specified algorithm.

5.2. OPL Election

- runElection()

  This function is the driving force of the OPL Election class. It will call all of the functions for the algorithm of OPL voting. This function will return -1 if there has been an error, and will otherwise return 0 if everything operated properly. It will also close the media, audit, and CSV files at the end of the function, which will be the end of this election process.

- readOPLCSV()

  This function will read in the first part of the OPL CSV file. It will set the variables totalNumBallots, totalNumSeats, and numSeatsLeft that are in the OPL Election class. The function will read this information directly from the CSV, and numSeatsLeft will not be read but will be set equal to totalNumSeats initially.

  It will also set the quota for the election, which is calculated by dividing the total number of votes by the number of seats in the election. This result will be put in the quota variable in the OPL Election class.

  The function will also create the party objects for each party that is read in from the CSV. As each party is read, an object will be created until there are no more parties/candidates to read in. This will also put each party into the party array list in this class.

  Additionally, this function will create candidate objects and assign them to their respective parties. It will read in the candidate name from the CSV, and assign it to the party class that is specified.

- readBallots()

  This function will create ballot objects for each individual ballot that is read in from the CSV. The function will read one ballot at a time, create a ballot object for each, and will assign them a unique ID number in the object, which will be stored in the privateID object for each ballot.

  It will also distribute each ballot that is touched to their respective candidate that they voted for, which will be stored in the cBallots object in the candidate class.

- printToScreen()

  This function will print information to the screen at the end of the program, giving the general election info to the user. This information will include the party, the number of seats that each party was given, and which candidates got the seats for each party.

- writeToMediaFile()

  This function will write some election information to the specified media file at the end of the election. This information will include the winner(s) of the election, the candidates and what percentage of votes went to each candidate. In this case, there are no specified winners, so it will store information for each party that obtained seats.

- writeToAuditFile()

  This function will write all of election information to the specified audit file at the end of the election. This information will include the basic election data, election results, as

was put in the media file, as well as unique ballot IDs, and a breakdown of how the ballots were distributed at each step of the analysis.

- allocateByQuota()
  This function will be used to allocate the first round of seats to each party. It will loop through each party in the party ArrayList. It will access the numBallots class variable for each party and divide it by the quota. It will take the quotient from this calculation and set the numSeats class variable for the party and it will take the remainder of this calculation and set the remainder class variable for the party.
- allocateByRemainder()
  This function will allocate the second round of seats for each party based on the remaining votes contained in the remainder object for each party in the election. Remaining seats will be allocated based on the parties that have the highest remainder. The party with the highest remainder will be given a seat first, and then it moves down the list until there are no seats left to be allocated.
- partyNumBallots()
  This function will iterate through the party array list object in this class. As it is iterating through, it will call the calculateNumBallots() function for each party, which will calculate the number of ballots that each party has.
  As it is iterating through the party list, it will also call sortCandidate() function, which will sort the candidates in each party, with the person who has received the most votes at the beginning of the array.
  This function will be called after all objects have been created and assigned.

5.3. IR Election

- runElection() : int
  This function is the driving force for the IR election class. It will call any necessary functions to run the IR algorithm. This function will return -1 if there has been an error, and will otherwise return 0 if no errors were encountered.
- readIRCSV()
  This function will read lines 2-4 in a CSV file that has been determined to be an IR CSV file. It will set the variable totalNumBallots and initialize the candidates ArrayList to be the size read from line 4. In addition, it will create and add candidate objects to this candidate's ArrayList.
- readBallots()
  This function will read the majority of the input CSV file. For each line of input, an IRBallot object is created with a unique ID, the ranking ArrayList is initialized to the size of the candidate's ArrayList, and the rankIndex is set to 1. This function will then loop through the ballot's ranking of the candidates and place each candidate in the appropriate spot in the ranking ArrayList.
- findMajority() : Candidate
  This function will use the variables totalNumBallots and cNumBallots to calculate a percentage of the total vote each candidate has. Once the majority has been found, the function will return the Candidate object that corresponds to that majority.
- findLeastCand() : Candidate

This function will loop through the currCandidates ArrayList and calculate the minimum number of votes a candidate has. While looping, if a tie for minimum number of votes is found, the coinToss() function is called to determine which candidate continues as the minimum. Once the ArrayList has been completely looped through, the Candidate that was determined to have the least amount of votes is returned.

- redistributeBallots()
  This function will call findLeastCand() to determine which candidate to move to from the currCandidates ArrayList to the eliminatedCandidates ArrayList. The function will then also use this returned candidate object to redistribute the ballots in the cBallots ArrayList. If the rankIndex of the ballot is less than the number of candidates (and not equal to -1) and the candidate at rankIndex is determined to still be in the currCandidates ArrayList, the ballot can be added to the cBallots ArrayList of the new candidate. If both of these conditions are not met, then the rankIndex will be set to -1 and the ballot will stay assigned to the eliminated candidate.
- writeToMediaFile()
  This function will write some election information to the specified media file at the end of the election. This information will include the winner of the election, the candidates and what percentage of votes went to each candidate.
- writeToAuditFile()
  This function will write final election information to the specified audit file. This information will include the basic election data, election results, as was put in the media file, as well as unique ballot IDs, and a breakdown of how the ballots were distributed at the end of the analysis.
- writeToAuditFile(Candidate c)
  This function will write election information pertaining to an eliminated candidate, which is passed in. This information will include the percentage of votes and the ballots that the candidate had before they were eliminated. It is assumed that this function will be called when it is determined that a candidate will be eliminated, but before the ballots assigned to this candidate are redistributed.
- printToScreen()
  This function will print information to the screen at the end of the program, giving the general election info to the user. This information will include the party, the number of seats that each party was given, and which candidates got the seats for each party.

5.4. Party

- sortCandidates()
  This function will be used to sort the candidates ArrayList. It will access the number of ballots for each candidate in the ArrayList and put the candidate with the most ballots in the 0th position and the candidate with the least ballots in the last position. It will call the coinToss() to handle any ties.
- calculateNumBallots()
  This function will be used to calculate the total number of ballots for the party. It will go through the candidates ArrayList and sum the number of ballots for each candidate. It will then set the class variable pNumBallots to this sum.
- addCandidate(Candidate)

This function will be used to add individual candidates to the candidates ArrayList. It takes a Candidate object as a parameter and adds that candidate to the ArrayList.

- CoinToss(int num) : int

  This function will be used to break any ties throughout either algorithm. It takes in the number of candidates involved in the tie, and returns an integer representing the losing candidate. It will simulate 1000 coin tosses and take the 1001 result to ensure randomness.

5.5. Candidate

- addBallot(Ballot)

  This function will be used to add individual ballots to the cBallot ArrayList. It takes a Ballot object as a parameter and adds that ballot to the ArrayList.

5.6. Ballot

The ballot class and its subclasses, OPLBallot and IRBallot, do not contain any object methods.

6. Human Interface Design

6.1. Overview of User Interface

The interface of the voting system is simple. There are no special graphical interfaces. The user will interact with the program through a computer terminal. The system will prompt the user for the CSV file, audit file, and media file names individually. Election results will be printed to the terminal at the end of the program. These will be the only interaction between the user and the system. Refer to section 6.2 for examples of user and terminal interaction. Once the program has completed, the audit and media files can be found in the same directory as the program in .txt format.

6.2. Screen Images

Display screenshots showing the interface from the user's perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

Below are example interactions between the user and the system during the program. These interactions will only occur at the beginning and end of the system.

1. This is a welcome message displayed to the user. The system prompts the user for a CSV file name.

```
Please enter the name of your file (do not include .txt extension):
OPLTest
```

2. The system asks the user what to name the media file and the audit file. If nothing is entered a program generated default name will be used.

```
What do you want the name of the audit file to be?
Do not include any extensions, the system will produce a .txt file
If you wish to use the default naming conventions, enter default (D)
aud
You have entered: aud
 Is this correct? (Y/N)
N
Enter the correct name:
audit_1

What do you want the name of the media file to be?
Do not include any extensions, the system will produce a .txt file
If you wish to use the default naming conventions, enter default (D)
med
You have entered: med
 Is this correct? (Y/N)
N
Enter the correct name:
media_1
```

3. The system displays election results and general information to the user through the terminal when the algorithm completes.

```
------------------------------
Election Results
------------------------------
D
  Number of Seats Won: 2
  Candidates filling seats:
     Pike
     Foster
------------------------------
R
  Number of Seats Won: 1
  Candidates filling seats:
     Borg
------------------------------
I
  Number of Seats Won: 0
  Candidates filling seats:
------------------------------
```

6.3. Screen Objects and Actions

There are no screen objects used in this system. The terminal handles all interaction with the user. Any prompts by the system will be text only and it will expect text only as a response from the user.

7. Requirements Matrix

Below is a table containing all of the functional requirements in the SRS and the component(s) which fulfill each requirement.

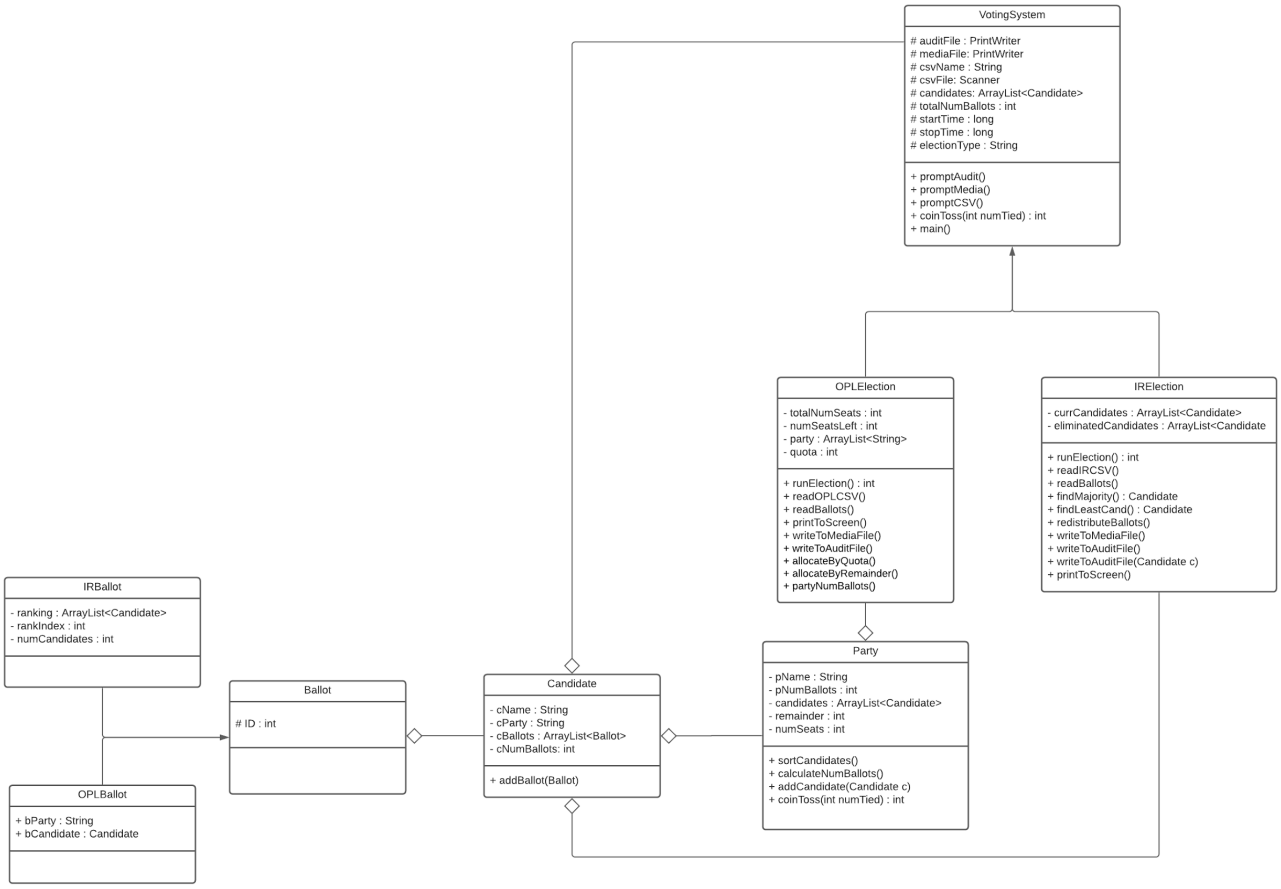
Requirement	Component
The system must prompt the user for the audit filename. If no name is provided a default name will be used	promptAudit() in VotingSystem class
The system must keep track of and store each ballot's path throughout the algorithm.	readBallots() in OPLElection class and the IRElection class Creates ballot objects of the Ballot class
The system must create an audit file with the name provided by the user or the default name.	promptAudit() in VotingSystem class
The system must write all of the election information to the audit file: The type of election, the number of candidates, the candidates' names, the candidates' parties, the percentage of votes awarded to each candidate, and the path of each ballot.	writeToAuditFile() in IRElection and OLPElection classes
The system must prompt the user for the media file name. If no name is provided a default name will be used.	promptMedia() in VotingSystem class
The system must create a media file with the name provided by the user or the default name.	promptMedia() in VotingSystem class
The system must write the election information that can be released to the public to the media file: The type of election, the number of candidates, the candidates' names, the candidates' parties, the percentage of votes awarded to each candidate, and the winner(s) of the election.	writeToMediaFile() in IRElection and OLPElection classes

System must read the first line of the CSV file to determine the type of election, either IR or OPL, then will use it to run the corresponding system within the program.	promptCSV() in VotingSystem class
The Instant Runoff System will determine the winner according to each ballot's ranking of each candidate. It is important that the system keep track of the path of each ballot. The system will analyze each ballot, eliminate the candidates with the lowest rankings, and redistribute votes until there are definite winner(s). Implementation details are TBD.	runElection() in IRElection class
The Open Party List System will determine the winner according to the candidate that each ballot votes for. The votes will go to the corresponding party which will then determine how many seats each gets. Implementation details are TBD.	runElection() in OPLElection class
The system will store the winner and calculated information for later use in the program.	Data is stored in the Party, Candidate, and Ballot (IRBallot and OPLBallot) classes
The program will prompt the user for the file name.	promptCSV() in the VotingSystem class
The program will open and read the file to analyze the election.	promptCSV() in the VotingSystem class readOPLCSV() in the OPLElection Class readIRCSV() in the IRElection Class
The program will store the data from the file to be referenced later.	promptCSV() will store the initial data from the file readOPLCSV() will store the specific election information from the file for OPL readIRCSV() will store the specific election information from the file for IR
The CSV file is in the format specified in section 6.1, CSV File Requirements	promptCSV() in VotingSystem, readOPLCSV, readBallots() in OPLElection, and readIRCSV(), readBallots() in IRElection will throw an exception if this is not true.
The program will display the type of election.	printToScreen() in OPLElection and IRElection
The program will display candidate information, including name, party affiliation, and percentage of votes received.	printToScreen() in OPLElection and IRElection

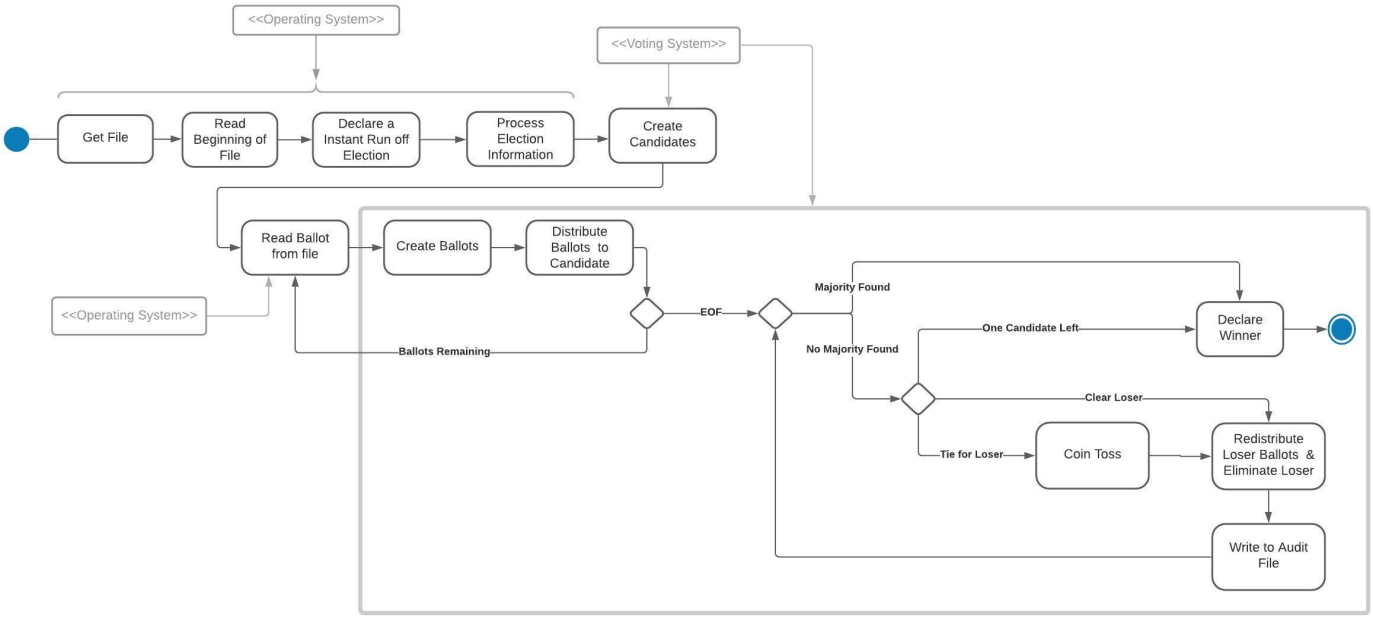
The program will display the number of candidates involved in the election and the number of seats up for election (if applicable to the type of election).	printToScreen() in OPLElection and IRElection
The program will display the number of ballots cast.	printToScreen() in OPLElection and IRElection
The program will display the determined winner(s) of the election.	printToScreen() in OPLElection and IRElection

8. Appendices

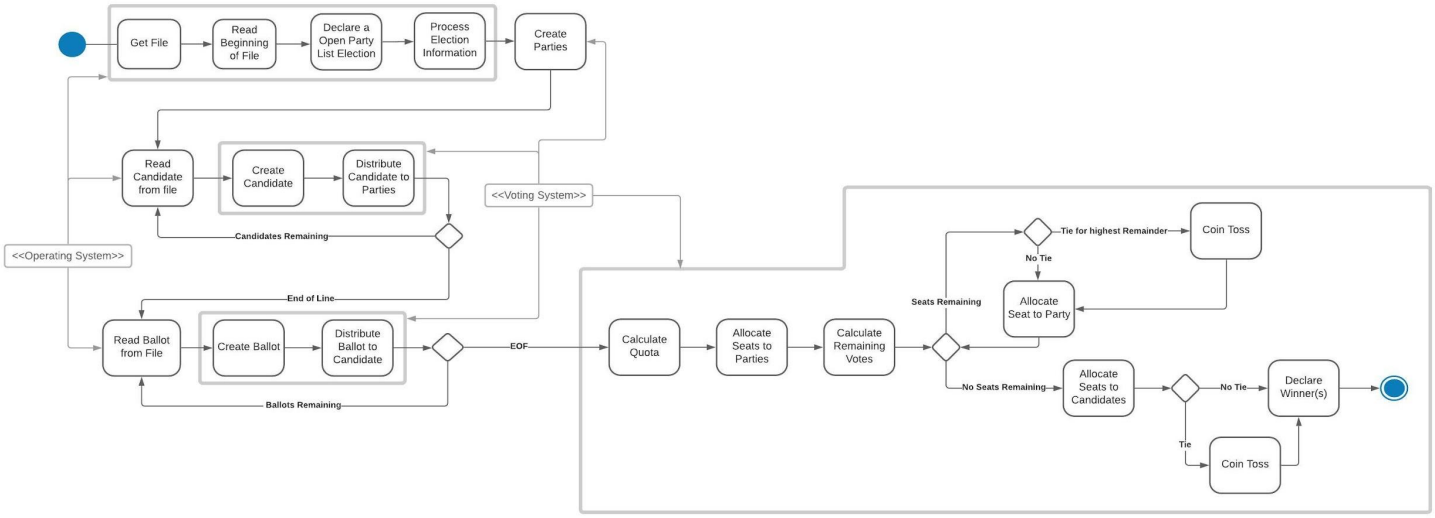
8.1. Appendix A: Class Diagram



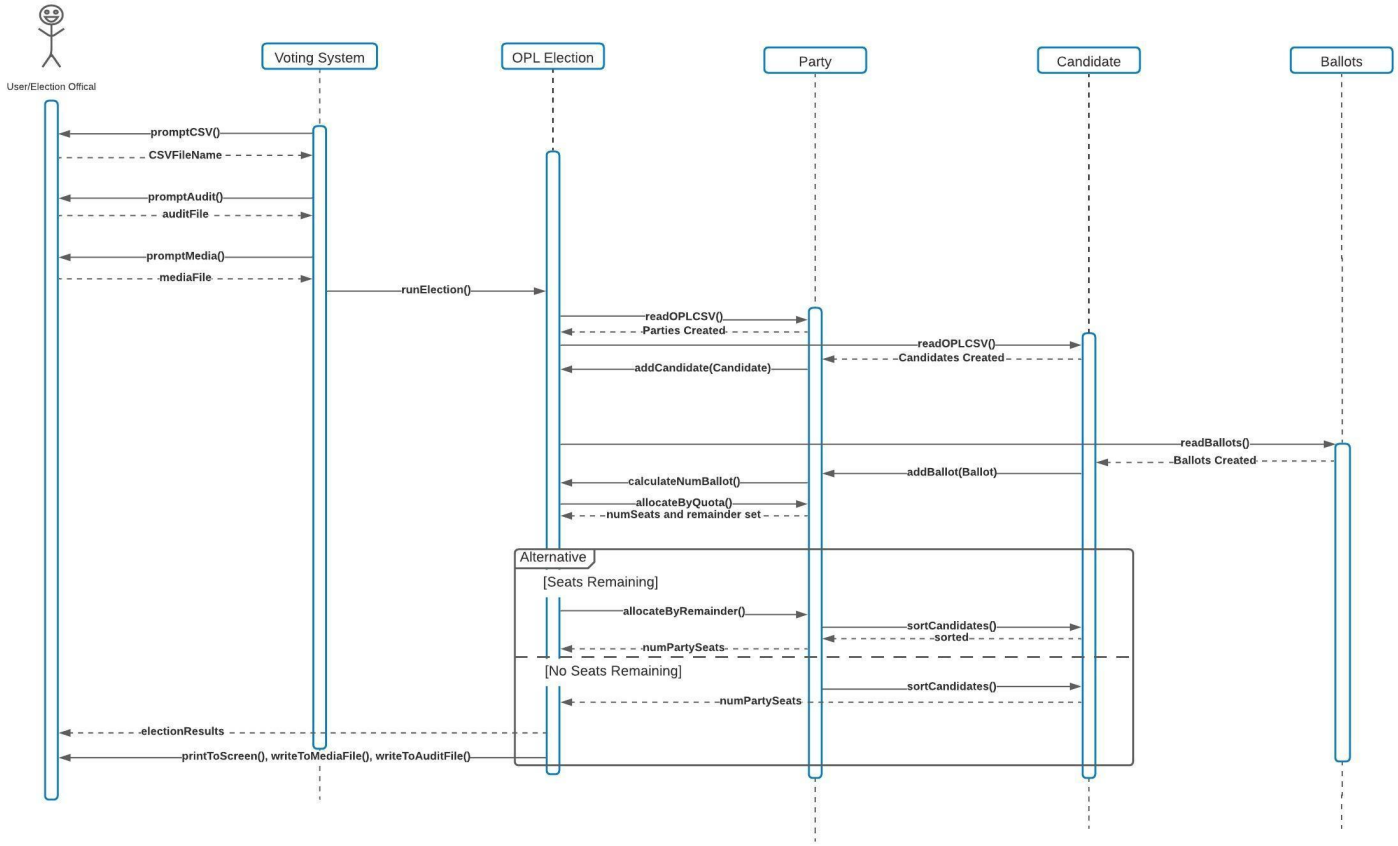
8.2. Appendix B: Instant Runoff Election Activity Diagram



8.3. Appendix C: Open Party List Election Activity Diagram



8.4. Appendix D: Open Party List Election Sequence Diagram



8.5. Appendix E: Use Case Document Reference

The use cases associated with this voting system can be found in a separate document labeled “UseCases_Team6”.