

Team 9

Vote Counting System

Software Design Document

Names: Arpita Dev, Neha Bhatia, Praful
Das, Stuti Arora

Date: 10/27/20

TABLE OF CONTENTS

1. INTRODUCTION	2
1.1 Purpose	2
1.2 Scope	2
1.3 Overview	2
1.4 Reference Material	2
1.5 Definitions and Acronyms	3
2. SYSTEM OVERVIEW	3
3. SYSTEM ARCHITECTURE	4
3.1 Architectural Design	4
3.2 Decomposition Description	7
3.3 Design Rationale	10
4. DATA DESIGN	11
4.1 Data Description	11
4.2 Data Dictionary	12
5. COMPONENT DESIGN	16
6. HUMAN INTERFACE DESIGN	35
6.1 Overview of User Interface	35
6.2 Screen Images	35
6.3 Screen Objects and Actions	40
7. REQUIREMENTS MATRIX	41
8. APPENDICES	50

1. INTRODUCTION

1.1 Purpose

The purpose of this Software Design Document (SDD) is to describe the architecture, system design, and design components of the Vote Counting System using the Waterfall method. This document is intended for developers, testers, project managers, and stakeholders involved in the development and deployment of the software. It aims to provide detailed technical guidance on how the system will be built, structured, and implemented, ensuring that all the requirements from the Software Requirements Specification (SRS) are addressed adequately.

1.2 Scope

This document covers the Waterfall Vote Counting System, a software application designed to count votes for both Instant Runoff (IR) and Open Party Listing (OPL) voting mechanisms, using the Waterfall method. The primary goals of this project are:

- To ensure accurate and efficient vote counting for both IR and OPL voting systems.
- To generate a tamper-proof audit report detailing the vote counting process and results.
- To maintain the integrity, safety, and security of the election data.

The benefits of this system include enhanced election transparency, reduced possibilities of fraud, and improved trust in the election process.

1.3 Overview

The SDD is structured to first introduce the readers to the project's purpose, scope, and essential definitions. It will then delve into detailed system architectures, data designs, interfaces, and components. Subsequent sections will address specific design considerations, constraints, and validation criteria, ensuring a comprehensive blueprint for the software's development.

1.4 Reference Material

- SRS Document: <https://github.umn.edu/umn-csci-5801-03-F23/repo-Team9/tree/main/SRS>

1.5 Definitions and Acronyms

- **SDD (Software Design Document):** A document detailing the architecture, system design, data design, and interface requirements of a software system.
- **SRS (Software Requirements Specification):** A document capturing all requirements for a system, serving as a basis for further software design and development.
- **IR (Instant Runoff):** A voting method where voters rank candidates in order of preference. The count process involves multiple rounds until a candidate receives a majority.
- **OPL (Open Party List):** A voting system where votes are cast for parties rather than individual candidates. Seats are allocated based on vote proportions.
- **CLI (Command Line Interface):** A user interface where interactions occur through command lines or terminal.
- **CSV (Comma-Separated Values):** A plain text file format used to store structured data, with rows and columns separated by commas or other specified delimiters.
- **IEEE (Institute of Electrical and Electronics Engineers):** A professional organization dedicated to advancing technology in fields like electrical engineering and computer science.

2. SYSTEM OVERVIEW

Our project specifies the software requirements for a new standalone vote counting system. The system will be able to calculate election results based on two distinct voting algorithms: Instant Runoff Voting (IRV) and Party List Voting using Open Party List (OPL). Its primary purpose will be to ensure efficient and accurate elections by processing electronic ballots, determining winners, and generating comprehensive audit reports. By providing a user-friendly and reliable platform for tallying votes, the software aims to uphold the integrity of elections and be a valuable tool for election officials compared to the previous method of hand-counting and tabulating votes. *Our design consists of a simple command-line interface for users that will only require a valid filename input.* Our application will then determine the election type, tabulate results, and generate the winner(s) and audit file.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The Vote Counting System is a critical component of the election process, connecting with three key systems: the Election Administration System, Voting System, and Ballot Validation & Conversion System. It plays a pivotal role in counting votes cast by voters. The system ensures the accuracy and integrity of the election results by processing and counting the received ballots, which are initially generated by the Voting System. It receives election parameters from the Election Administration System, and with the Ballot Validation & Conversion System verifies and converts the paper ballots into digital records.

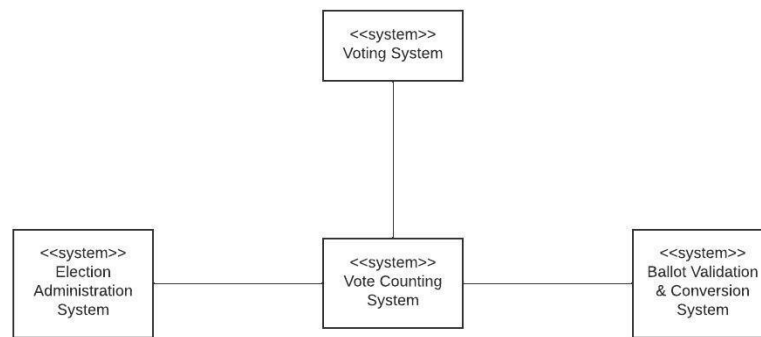


Figure 1. Context Model Diagram of how the Vote Counting System fits in the Election Process

The vote counting system will be designed with a modular program structure to ensure efficient and organized processing of election data. Each module, or subsystem, is responsible for specific tasks, and together, they achieve the complete functionality of the system. Here are the subsystems and how they collaborate to deliver the desired functionality:

Main Vote Counting System:

- Acts as the core component that orchestrates the entire vote counting process.
- Directs the flow of data and control to the relevant subsystems based on the election type and user commands.

Input System:

- Handles the collection and initial processing of input data, ensuring its validity and integrity.
- Passes processed data to other subsystems for further analysis.

Header Processing System:

- Identifies the type of election based on the header information from the input data.
- Extracts essential details like the number of candidates, party information, and ballot counts.

Ballot Data Processing System:

- Manages the processing and validation of ballot data.
- Distributes the ballots to the appropriate candidates or parties, preparing them for vote counting.

Candidate & Party Management System:

- Manages candidate and party information, allowing administrators to add, update, or remove candidates and parties.
- Ensures that candidate and party lists are up to date and used in the counting process.

Results Calculation System:

- Performs the actual vote counting, determining the winners and allocated seats.
- Adheres to the specific logic of each election type, whether it's Instant Runoff or Open Party List.

Audit File Creation System:

- Records and logs the entire vote counting process for transparency and auditing.
- Generates reports and audit files for users to verify the election results.

Testing System:

- Implements test cases and verification procedures to ensure the correctness of the counting process.
- Supports the testing and debugging of various components within the system.

User Interface System:

- Provides a user-friendly interface for interaction with the vote counting system.
- Allows users to initiate and monitor the counting process, making it accessible and intuitive.

These subsystems work in harmony to process election data, calculate results, and maintain transparency. The Main Vote Counting System serves as the conductor, directing data and control to the relevant subsystems based on the election type and user commands.

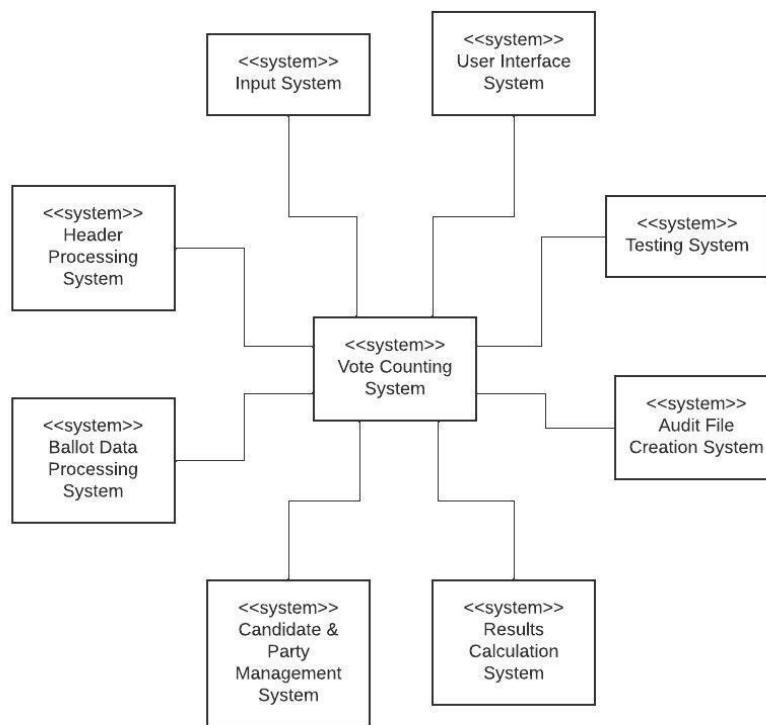


Figure 2. Context Model Diagram of the Vote Counting System

This modular structure allows for efficient development of our vote counting system, ensuring that it meets the requirements while providing a user-friendly and transparent

voting experience. The collaboration between these subsystems results in a well-organized and efficient program structure for accurate and reliable vote counting.

3.2 Decomposition Description

The central figure in our system is the Controller object, responsible for managing the main functionalities.

Controller: Our control center manages the main actions in the system. It holds references to a Test object for testing purposes and an InputFile object for file processing. The main() function initiates the system, while handleTest() is in charge when testing is required.

Test: The testing module provides extensive testing functions to ensure system accuracy. It checks file headers, audits, election processes, and results, improving reliability.

FileProcessor: Our file processor subsystem reads and processes election data. It interacts with the ElectionType object to understand the election type and manages the file's status.

ElectionType: The heart of our system, it's responsible for election-related tasks. It's an abstract class and manages ballot counts, candidate lists, parties, audit files, and more. IR and OPL election objects inherit from it.

OPElection: Represents Open Party List elections. It manages the number of seats and calculates election results, ensuring fairness and accuracy.

IRElection: Handles Instant Runoff elections. It efficiently manages candidate rankings and ensures a smooth voting process.

Candidate: These represent election candidates. They have names, vote counts, and party affiliations, which are integral to determining election results.

Party: The life of political parties in the system. They collect vote counts, list candidates, and contend for seats in OPL elections.

Ballot: The ballot objects process votes efficiently.

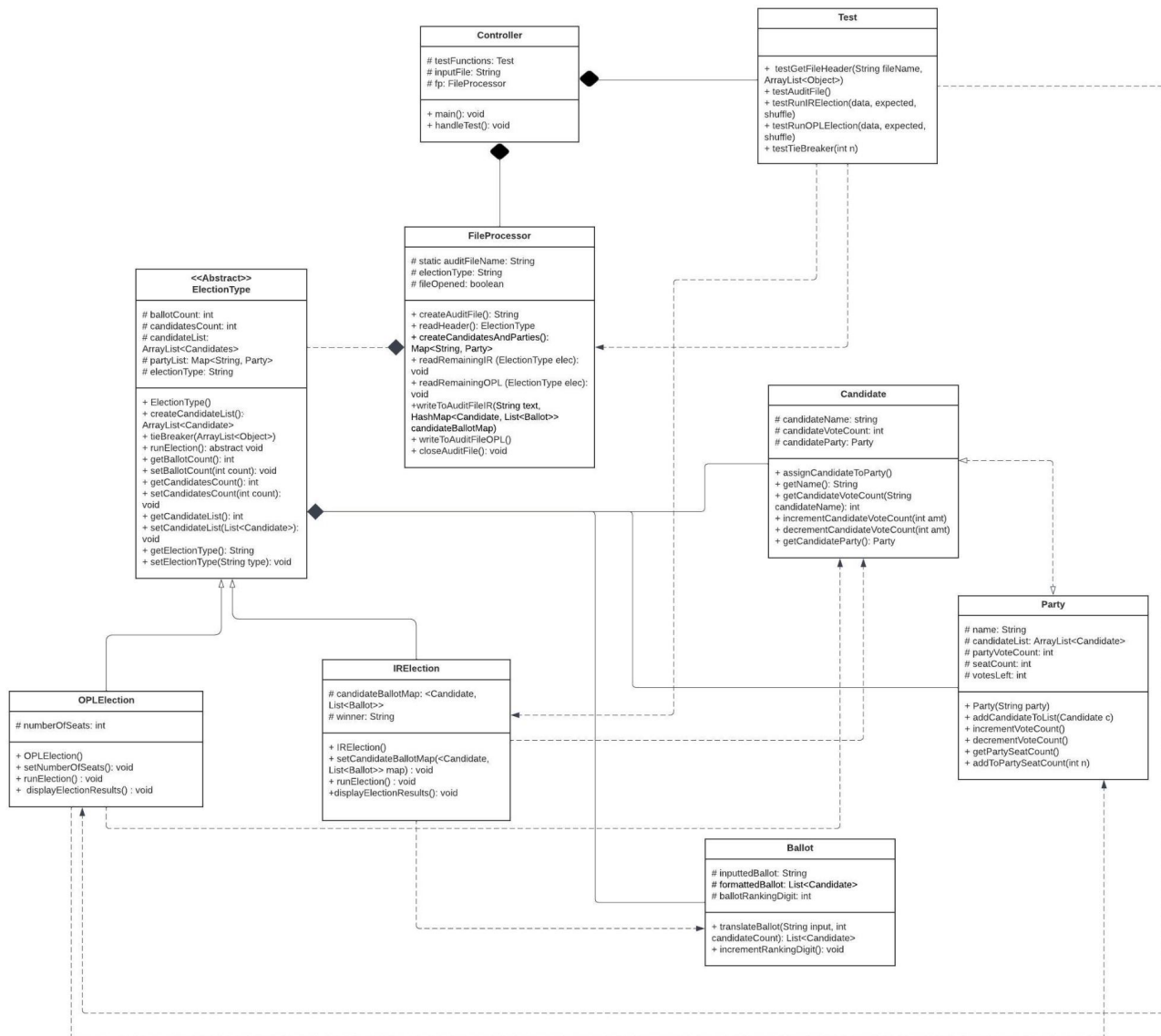


Figure 3. UML Class Diagram of the Vote Counting System

Instant Runoff (IR) election:

Figure 4. illustrates the process of conducting an Instant Runoff (IR) election, from its initiation to the declaration of the winner. It involves various subsystems, including the Election Administration System, Voting System, Ballot Validation and Conversion System,

and the Vote Counting System. In the beginning, the process involves declaration of candidates and finalizing the election date. Key activities include the creation of IR ballots, conducting the vote, validating and converting the ballots, and inputting them into the IR Vote Counting System. Finally, the system concurrently handles the declaration of the winner and the generation of an audit file.

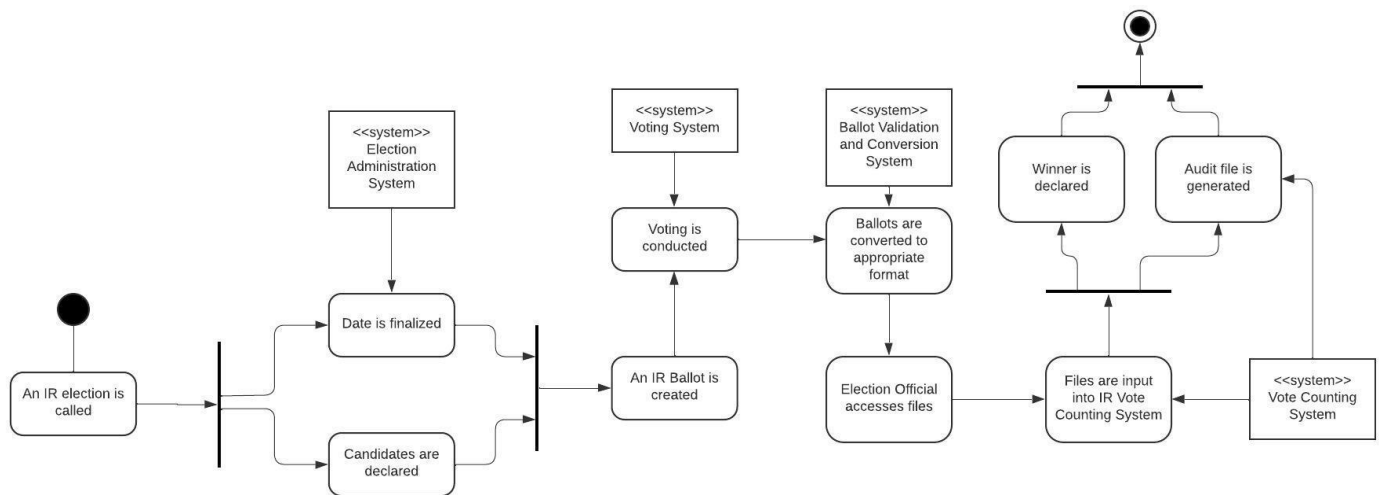


Figure 4. UML Activity Diagram of the IR Election Process

Open Party Listing (OPL) election:

Figure 5. outlines the process of handling an election data file, from input to declaring the winner(s). It involves objects such as Controller, FileProcessor, ElectionType, Party, Candidate and OPLElection. Key interactions include passing the file to the FileProcessor, which triggers actions like reading the header, creating an ElectionType, and reading the ballot. It also involves creating party list and candidate list, running the election, calculating the winner party, and writing to and saving the audit file. Finally, the results are displayed to the user.

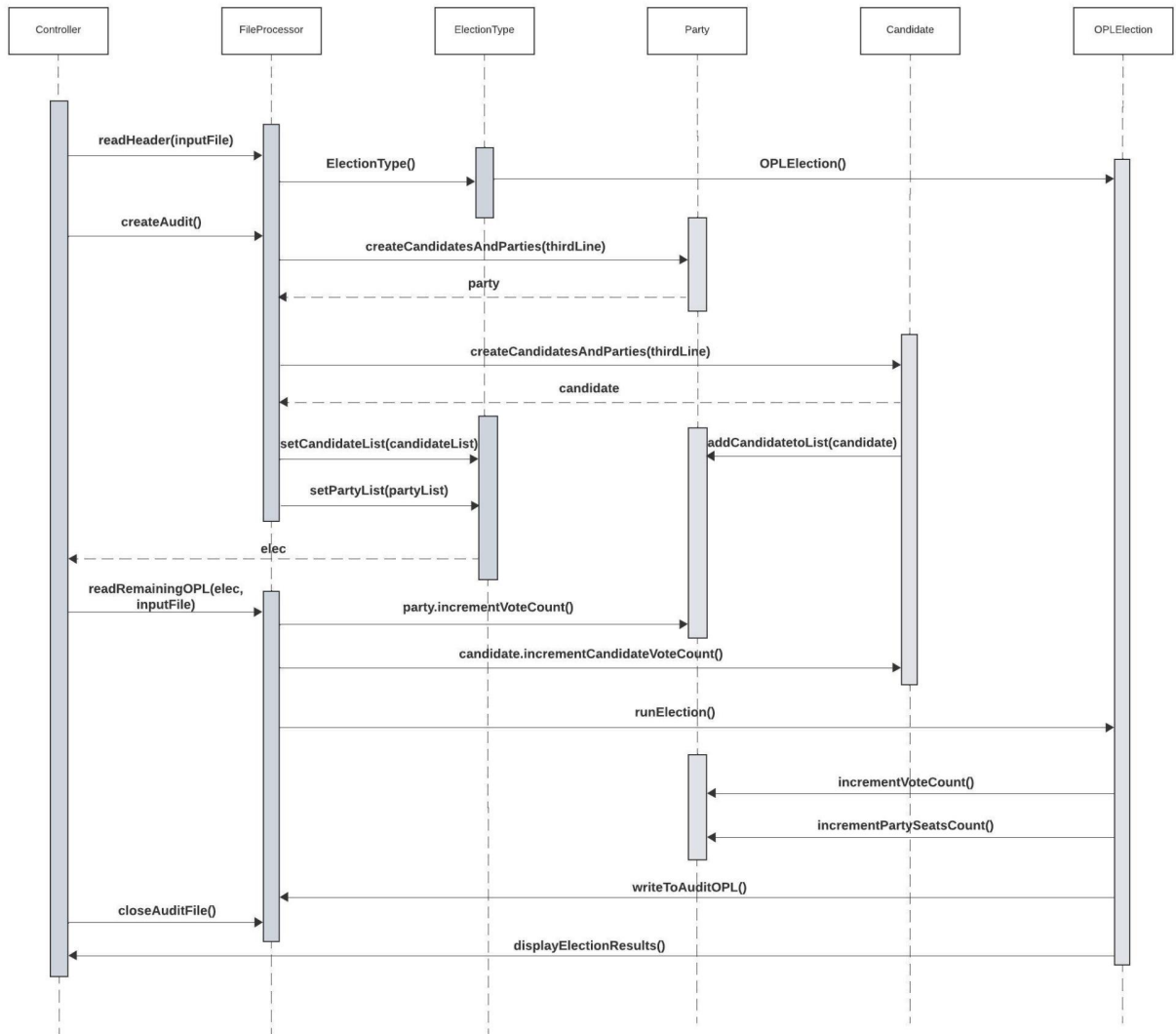


Figure 5. UML Sequence Diagram of the OPL Election Process

3.3 Design Rationale

Our choice of the modular program structure for the vote counting system was made after careful consideration of critical issues and trade-offs that ensure the system's efficiency and maintainability.

We selected a modular structure due to the following reasons:

Scalability and Maintainability: By breaking down the system into distinct subsystems, we ensure that each part is responsible for a specific function. This approach makes it easier to maintain and expand the system as requirements evolve. It also allows for parallel development, enhancing efficiency.

Flexibility: The modular design permits the easy replacement or upgrading of individual subsystems without affecting the entire system. This flexibility ensures adaptability to different election types and future changes in election rules and regulations.

Transparency and Auditability: The separation of tasks into subsystems promotes transparency and auditability. For instance, the Audit File Creation System records the entire vote counting process, allowing users to verify results, enhancing trust in the system.

While the modular program structure was chosen, we evaluated alternative architectures, such as a monolithic architecture.

In a monolithic architecture, the entire vote counting system would reside within a single application. However, we decided against it because it could lead to code complexity and difficulties in maintenance and scalability. A modular structure allows for clearer separation of concerns and more straightforward debugging.

In summary, the selected modular program structure offers the right balance between maintainability, transparency, and efficiency. It enables us to meet the requirements of the vote counting system while allowing for future adaptability and expansion as needed. This approach ensures that the system remains user-friendly and reliable while delivering accurate election results.

4. DATA DESIGN

4.1 Data Description

We chose to utilize Object Oriented programming for our system design and that makes up a majority of the data structure. Data parsed from the input file is stored locally in ArrayLists, Hashmaps and other variable types inside objects. The data is only stored while the program is running since we do not use any persisting data structures like a database. The only data that persists outside the local objects and variables in the program belongs to the audit file. The data in this file is written from both the IR and OPL Election classes and comes from changing values of variables while the election is being processed.

4.2 Data Dictionary

Ballot:

- Attributes:
 - inputtedBallot: String
 - formattedBallot: int[]
- Methods + Method Parameters:
 - translateBallot(String input)

Candidate:

- Attributes:
 - candidateName: String
 - candidateVoteCount: int
 - candidateParty: Party
- Methods + Method Parameters:
 - assignCandidateToParty()
 - getName()
 - getCandidateVoteCount(String candidateName)
 - incrementCandidateVoteCount()
 - decrementCandidateVoteCount()
 - getCandidateParty()

Controller:

- Attributes:
 - testFunctions: Test
 - inputFile: FileProcessor

- Methods + Method Parameters:

- main
- handleTest()

ElectionType:

- Attributes:

- auditFileName: String
- ballotCount: int
- candidatesCount: int
- candidateList: ArrayList<Candidates>
- electionType: String
- partyList: Map<String, Party>

- Methods + Method Parameters:

- ElectionType()
- ElectionType(int ballotCount, int candidateCount)
- createCandidateList():
- tieBreaker(ArrayList<Object>)
- ***runElection()***
- *getBallotCount()*
- *setBallotCount()*
- *getCandidatesCount()*
- *setCandidatesCount()*
- *getCandidateList()*
- *setCandidateList()*

- *getElectionType()*

FileProcessor:

- Attributes:
 - fileName: String
 - electionType: String
 - fileOpened: boolean
- Methods + Method Parameters:
 - readHeader()
 - readRemainingIR (ElectionType elec)
 - readRemainingOPL (ElectionType elec)
 - createAuditFile()

IRElection:

- Attributes:
 - candidateBallotMap: <Candidate, List<Array>>
 - winner: String
- Methods + Method Parameters:
 - IRElection()
 - writeToAuditFile()
 - createCandidateBallotMap()
 - runElection()
 - displayElectionResults(String electionType, int ballotCount, String winner, <candidate breakdown>, <audit-file-name>)
 - writeToAuditFileIR(HashMap<Candidate,List<Ballot>>, candidateBallotMap, String text)

OPElection:

- Attributes:
 - numberOfSeats: int
- Methods + Method Parameters:
 - OPElection()
 - writeToAuditFile()
 - setNumberOfSeats()
 - runElection()
 - displayElectionResults(String electionType, int numberOfSeats, int ballotCount, ArrayList<Candidate> candidateWinners, **<candidate breakdown>**, auditFileName)

Party:

- Attributes:
 - name: String
 - candidateList: ArrayList<Candidate>
 - partyVoteCount: int
 - seatCount: int
 - votesLeft: int
- Methods + Method Parameters:
 - Party(String party)
 - addCandidateToList(Candidate c)
 - incrementVoteCount()
 - decrementVoteCount()

Test:

- Attributes:
- Methods + Method Parameters:
 - testFileHeader(String fileName, ArrayList<Object>)
 - testCreateAuditFile(file, expected)
 - testWriteToAuditFile(file, expected)
 - testRunIRElection(data, expected, shuffle)
 - testRunOPLElection(data, expected, shuffle)
 - testDisplayElectionResultsOPL(data, expected)
 - testTieBreaker(int n)

5. COMPONENT DESIGN

Controller:

main()	<p>DISPLAY "Welcome to Election Counter!"</p> <p>DISPLAY "List of commands:"</p> <p>DISPLAY "<yourfilename>.csv: To count votes for Instant Runoff or Open Party List voting, enter the full filename with the ".csv" extension."</p> <p>DISPLAY "test: To access the testing environment"</p> <p>DISPLAY "^C: Quit the program"</p> <p>DISPLAY "Enter your command:"</p> <p>SET userInput = Read user input from terminal</p> <p>IF userInput is "test" Then</p> <p style="padding-left: 40px;">CALL handleTest()</p>
--------	---

	<pre>ELSE IF userInput ends with ".csv" THEN CREATE a new FileProcessor object SET FileProcessor.inputFile = userInput IF file found and opened successfully THEN SET FileProcessor.fileOpened = True SET FileProcessor.electionType = readHeader(userInput) IF FileProcessor.electionType.getElectionType() = "IR" THEN readRemainingIR(FileProcessor.electionType, userInput) ELSE readRemainingOPL(FileProcessor.electionType, userInput) ELSE SET FileProcessor.fileOpened = False END IF ELSE IF userInput is "exit" THEN exit the system ELSE //user input is invalid PROMPT user for valid input END IF</pre>
--	--

handleTest()	CALL testFunctions.testFileHeader(file, expected) CALL testFunctions.testAuditFile() CALL testFunctions.testRunIRElection(data, expected, shuffle) CALL testFunctions.testRunOPLElection(data, expected, shuffle) CALL testTieBreaker(int n)
--------------	---

FileProcessor:

readHeader(string inputFile)	FirstLine = Read the first line of the input file elec = CREATE new ElectionType IF FirstLine is "IR" THEN ElectionType elec = CREATE new IRElection() SET elec.electionType = "IR" SET elec.candidatesCount = Read the second line of the input file as an integer CREATE thirdLine = Read the third line of the input file SET elec.partyList = createCandidatesAndParties(thirdLine) SET elec.ballotCount = Read the fourth line of the input file as an integer ELSE IF FirstLine is "OPL" Then ElectionType elec = CREATE new OPLElection() SET elec.electionType = "OPL" SET elec.candidatesCount = Read the second line of the input file
------------------------------	--

	<p>as an integer</p> <p>CREATE thirdLine = Read the third line of the input file</p> <p>SET elec.partyList = createCandidatesAndParties(thirdLine)</p> <p>SET elec.numberofSeats = Read the fourth line of the input file as an integer</p> <p>SET elec.ballotCount = Read the fifth line of the input file as an integer</p> <p>END IF</p> <p>RETURN elec</p>
createCandidates AndParties(String line, ElectionType elec)	<p>// for reading and processing the third line (candidates and parties)</p> <p>partyList = CREATE a new HashMap<String, Party></p> <p>candidateList = CREATE a new ArrayList<Candidates></p> <p>LOOP FOR each candidate IN the third line:</p> <p>SET name = EXTRACT candidate name from the candidate</p> <p>SET party = EXTRACT party name from the candidate</p> <p>IF partyList.containsKey(party) Then</p> <p>partyObject = partyList.get(party)</p> <p>ELSE</p> <p>partyObject = CREATE a new Party object with the party name</p> <p>partyList.put(party, partyObject)</p> <p>END IF</p>

	<p>CREATE a Candidate object with the name and partyObject</p> <p>ADD the Candidate to partyObject.CandidateList</p> <p>ADD the Candidate to candidateList</p> <p>SET elec.candidateList to candidateList</p> <p>SET elec.partyList to partiesCreated</p>
readRemainingIR(ElectionType elec, string inputFile)	<p>candidateBallotMap = CREATE a new empty dictionary <Candidate, List<Ballot>></p> <p>LOOP from 4th line of input file until the end of the file is reached:</p> <p> Read a line from the input file</p> <p> CREATE translatedBallot = translateBallot(line, elec.candidatesCount)</p> <p> SET topCandidate = Get the first element of the newBallot array</p> <p> IF candidateBallotMap.containsKey(topCandidate) THEN</p> <p> // Add the ballot to the topCandidate's list of ballots</p> <p> APPEND translatedBallot to candidateBallotMap.get(topCandidate)</p> <p> ELSE</p> <p> // Create a new entry for topCandidate with an empty list of ballots</p> <p> SET candidateBallotMap entry for topCandidate to empty list of ballots</p> <p> END IF</p> <p>END LOOP</p>

	SET elec.candidateBallotMap = candidateBallotMap
readRemainingOP L (ElectionType elec, string inputFile)	CREATE candidateList = elec.getCandidateList() LOOP from 6th line until the end of the input file is reached: Read a line from the input file CREATE candidateIndex = Parse the line to obtain the the position of "1" CREATE candidate = candidateList[candidateIndex] CREATE party = candidate.getCandidateParty() party.incrementVoteCount() candidate.incrementCandidateVoteCount() END LOOP
writeToAuditFileI R(HashMap<Candidate, List<Ballot>> candidateBallotM ap, String text)	OPEN file (if not already open) WRITE newline and text to file FOR each candidate in candidateBallotMap: WRITE "Candidate " + Candidate.name WRITE "Associated Ballots" + map.get(candidate) CLOSE file
writeToAuditFileO PL(String round, ArrayList<Party> partyList, Hashmap<Integer	OPEN file Add a newline and print the String round to the file WRITE "Total Seats:" + numberOfSeats

<pre>, ArrayList<Party>> votesLeftDict, ArrayList<<Party, ArrayList<Party>> tieBreaks, int seatsLeft)</pre>	<pre>WRITE "Current Seats Left:" + seats left WRITE "Seat Quota:" + floor(ballotCount / numberOfSeats) WRITE "Party Name Total Votes Seats Assigned Remainder Votes Candidates (Votes)" FOR each Party in PartyList: WRITE Party.name, Party.voteCount, Party.getPartySeatCount(), party.getVotesLeft() IF tieBreaks is NOT empty: WRITE "Tie Breaks" WRITE "Winner:" + tieBreaks.Party WRITE "Parties that had ties:" + tieBreaks.PartyList</pre>
<pre>closeAuditFile(): void</pre>	<pre>CLOSE the audit file if it's currently open</pre>

Party:

<pre>Party(String party)</pre>	<pre>// Validate if partyName is not empty or null, to ensure every party has a valid name. IF partyName is not empty and partyName is not null THEN SET name to partyName // Assign the provided party name to the name attribute ELSE THROW error "Party name cannot be empty or null."</pre>
--------------------------------	--

	<p>END IF</p> <p>// Prepare a list to store the candidates that will be associated with this party.</p> <p>INITIALIZE candidateList as empty ArrayList<Candidate></p> <p>SET partyVoteCount to 0</p> <p>SET seatCount to 0</p> <p>SET votesLeft to 0</p>
addCandidateToLi st(Candidate c)	<p>// Validate if the candidate is not already in the list to avoid duplicates</p> <p>IF c is not in candidateList THEN</p> <p> APPEND c to candidateList</p> <p>// Insert the new candidate to the party's list</p> <p>ELSE</p> <p> DISPLAY "Candidate is already in the list."</p> <p>END IF</p>
incrementVoteCo unt()	<p>INCREMENT partyVoteCount by 1</p>
decrementVoteCo unt()	<p>// Ensure vote count doesn't go negative</p> <p>IF partyVoteCount is greater than 0 THEN</p> <p> // Decrease the vote count of the party by one; this could be used in scenarios where a vote is invalidated or retracted</p>

	<p>DECREMENT partyVoteCount by 1</p> <p>ELSE</p> <p>DISPLAY "Vote count is already at the minimum value of 0."</p> <p>END IF</p>
getPartySeatCount()	RETURN seatCount
addToPartySeatCount(int n)	<p>// Validate the number to be added.</p> <p>IF n is greater than 0 THEN</p> <p> // Add n to the seat count.</p> <p> ADD n to seatCount</p> <p>ELSE</p> <p> DISPLAY "The number to be added should be positive."</p> <p>END IF</p>

ElectionType:

ElectionType():	<p>Set ballotCount to 0</p> <p>SET candidateCount to 0</p> <p>SET candidateList to new ArrayList<Candidate></p> <p>SET electionType to an empty string</p> <p>SET numberOfSeats to 0</p>
-----------------	--

ElectionType(int ballotCount, int candidateCount):	SET ballotCount to ballotCount SET candidateCount to candidateCount SET candidateList to createCandidateList() SET electionType to an empty string SET numberOfSeats to 0
createCandidateList(string)	CREATE an empty ArrayList<Candidate> SPLIT the string into individual candidate entries using a delimiter (e.g., ',') CREATE an empty HashMap<String, Party> to store parties created FOR each candidate entry in the split string: EXTRACT candidate name and party from the candidate entry SET name to the extracted candidate name SET party to the extracted party name IF partiesCreated.containsKey(party): RETRIEVE the partyObject from partiesCreated using the party name ELSE: CREATE a new Party object with the party name ADD the partyObject to partiesCreated with the party name as the key CREATE a Candidate object with the name and the partyObject ADD the created Candidate to the Party's candidate list

	RETURN the ArrayList<Candidate> containing the populated candidate list
tieBreaker(ArrayList<Object> objects)	<p>IF objects is empty OR objects has only one object:</p> <p>RETURN the first object in the list (no tie to break)</p> <p>ELSE IF objects has more than one object:</p> <p>GENERATE a random index (e.g., random integer between 0 and the size of the objects list)</p> <p>RETURN the object at the randomly generated index as the winner</p> <p>RETURN null (no winner determined by the random selection)</p>
getBallotCount()	RETURN ballotCount
setBallotCount(int count)	SET ballotCount to count
getCandidatesCount()	RETURN candidateCount
setCandidatesCount(int count):	SET candidateCount to count
getCandidateList()	RETURN candidateList
setCandidateList(ArrayList<Candida	SET candidateList to list

te> list):	
getElectionType()	RETURN electionType
setElectionType(s tring type)	SET electionType to type

Ballot:

translateBallot(Str ing line, int candidateCount)	<pre> // Split the input line by commas to obtain candidate rankings SPLIT the line into an array called rawRankings by using the ',' delimiter CREATE a new list called translatedBallot // Initialize a list of integers to represent rankings CREATE a new list of integers called rankings // Initialize a dictionary to map rankings to candidates CREATE a new dictionary called rankingToCandidate // Iterate through the rawRankings array FOR i = 0 to candidatesCount - 1: // Initialize the ranking to a default value if not specified in the rawRankings IF i < LENGTH(rawRankings) AND rawRankings[i] is not empty: SET ranking = CONVERT rawRankings[i] to an integer ELSE: </pre>
---	---

	<p>SET ranking = i + 1 // Default ranking</p> <p>ADD ranking to rankings</p> <p>// Map the ranking to the candidate</p> <p>SET candidate = Get the Candidate associated with ranking from elec.candidateList</p> <p>SET rankingToCandidate[ranking] = candidate</p> <p>// Map the rankings to candidates in the translatedBallot</p> <p>FOR each ranking in rankings:</p> <p>ADD rankingToCandidate[ranking] to translatedBallot</p> <p>RETURN translatedBallot</p>
incrementRankingDigit()	<p>GET ballot.ballotRankingDigit</p> <p>INCREMENT by one</p>

IRElection:

IRElection()	SET winner = ""
setCandidateBallotMap(HashMap<Candidate, List<Ballot>> mapPassedIn)	SET this.candidateBallotMap = mapPassedIn

runElection()	<p>SET boolean winnerFound to false;</p> <p>SET integer smallest to Integer.Maximum_Value</p> <p>while(!winnerFound)</p> <p> CREATE list of candidates called leastVotedCandidates</p> <p> FOR each candidate in candidateBallotMap.keySet:</p> <p> SET size = get the size of the list of ballots for each candidate</p> <p> CHECK if any candidate has more than 50%</p> <p> IF yes, flag = true, fill in winner, write to audit file + leave loop</p> <p> IF (size <= smallest)</p> <p> add to list</p> <p> GET candidateToBeRemoved from tie breaker function amongst list of leastVotedCandidates</p> <p> CALL writeToAuditFile("tie between candidates", candidateToBeRemoved.name + "randomly chosen to be removed", candidateBallotMap)</p> <p> GET list of ballots associated with candidateToBeRemoved from candidateBallotMap</p> <p> FOR each ballot in list</p> <p> UPDATE ballot ranking digit by 1</p> <p> GET candidate associated to ballot ranking digit (use list.indexOf())</p> <p> CHECK if candidate still exists in candidateBallotMap (have to grab candidate name from candidateList + then check that with map -</p>
---------------	--

	<p>need to have check to see if int is greeter than candidateCount)</p> <p>MOVE ballot to new top candidate in candidateBallotMap</p> <p>CALL FILEProcessor.writeToAuditFile("ballot shuffled", candidateBallotMap)</p> <p>this.displayIRElectionResults()</p>
displayElectionResults()	<p>DISPLAY "IR ELECTION"</p> <p>DISPLAY Winner</p> <p>FOR each candidate in candidateBallotMap</p> <p>WRITE "Candidate " + Candidate.name</p> <p>WRITE "Number of Votes " + candidateBallotMap.get(candidate).size()</p>

OPLElection:

OPLElection()	<p>SET numberOfSeats = 0</p> <p>SET partyList to new ArrayList<Party></p> <p>SET candidateList to new ArrayList<Candidate></p> <p>SET candidateCount to 0</p> <p>SET ballotCount to 0</p>
runElection()	<p>SET quota to floor(ballotCount / numberOfSeats)</p> <p>SET seatsLeft to numberOfSeats</p> <p>CREATE dictionary votesLeftDict that maps <Integer votesLeft,</p>

	<pre> ArrayList<Party>> FOR each Party value in keys of PartyList: DECREMENT seatsLeft by floor(Party.VoteCount / quota) CALL incrementPartySeats() with floor(Party.VoteCount / quota) ADD to Party to votesLeftDict at (PartyList.VotesLeft % quota) writeToAuditOPL("Round 1 Allocations", PartyList, votesLeftDict, <>, seatsLeft) IF seatsLeft > 0 FOR all the KEYS in votesLeftDict CREATE tieBreaks<Party, ArrayList<Party>> IF the value at votesLeftDict.key has more than one party: CALL tieBreaker with ALL the parties in the ArrayList SET curParty to tieBreaker winner ADD ArrayList and curParty to tieBreaks CALL curParty.incrementPartySeats with 1 SET curParty.VotesLeft to 0 CALL writeToAuditOPL("Round 2 Allocations" PartyList, votesLeftDict, tieBreaks, seatsLeft) ENDIF IF seats are still greater than 0, randomly assign seats amongst all the parties that have candidates > num of current party seats. CALL writeToAudit("Final Allocations", PartyList, votesLeftDict, <>, seatsLeft) ENDIF </pre>
--	--

displayElectionResults()	DISPLAY "OPL ELECTION" and other header stylistic stuff FOR each party in PartyList DISPLAY party.name, party.VoteCount, party.seatCount DISPLAY "PARTY WINNERS" FOR i = 0 to party.SeatCount DISPLAY party.CandidateList[i] DISPLAY party.CandidateList[i].voteCount
--------------------------	--

Test:

testGetFileHeader(String filename, ArrayList<Object> expected) ArrayList order: string ElectionType, numCandidates, string candidateList, int numBallots	CREATE a new fileProcessor obj called testFile using fname IF testFile.getFileOpened() is FALSE: CALL printTest("FILE - OPEN & HEADER", false, expected, emptyArrayList) RETURN CREATE a new ElectionType object et and set it to testFile.readHeader() IF et.getElectionType() equals "IR" AND et.getElectionType() equals expected[0] CAST et to type IElection CREATE a new ArrayList received with all attributes of the election type in the same order as expected adding numBallots ELSE CAST et to type OPLElection
---	--

	<p>CREATE a new ArrayList received with all attributes of the election type in the same order as expected adding numBallots</p> <p>FOR each value in expected and received</p> <p> IF expected is NOT equal to received</p> <p> CALL printTest with "FILE - OPEN & HEADER", false, expected, received</p> <p>CALL printTest with "FILE - OPEN & HEADER", true, expected, received</p>
testTieBreaker(int n)	<p>CREATE a new integer array called props of length n</p> <p>FOR i = 0, i < 5000, i++</p> <p> SET x to a Random number between 1 and N (inclusive)</p> <p> INCREMENT n[x]</p> <p>CALL printTest with "TieBreaker", an expected arraylist containing 100/n, a received arraylist containing counts of props/100</p>
printTest(String testName, boolean pass, ArrayList<Object> expected, ArrayList<Object> received)	<p>DISPLAY ("TEST - " + testName)</p> <p>IF pass is true</p> <p> DISPLAY ("TESTS PASSED")</p> <p>ELSE</p> <p> DISPLAY ("TESTS FAILED")</p> <p>DISPLAY all objects in expected</p> <p>DISPLAY all objects in received</p>

testAuditFile()	<p>CALL createAuditFile()</p> <p>RUN a mock election and CALL closeAuditFile()</p> <p>EXPECT file permissions for the generated Audit file to be read-only</p>
testRunIRElection(file, String expected)	<p>CALL fileProcessor with the file and call the electionObj.runElection()</p> <p>PARSE String expected to extract expected winner, other candidates and their corresponding vote percentages.</p> <p>SET testPassed to true</p> <p>Using JUnit -</p> <p>EXPECT winner to match winner in the expected string and the vote percentage to match as well, otherwise set testPassed to false</p> <p>EXPECT candidates and their vote percentages match, otherwise set testPassed to false</p> <p>CALL printTest("Run IR Election", testPassed, expected, electionObj)</p>
testRunOPLElection(file, String expected)	<p>CALL fileProcessor with the file and call the electionObj.runElection()</p> <p>PARSE String expected to extract expected party seats, and corresponding assigned candidates</p> <p>SET testPassed to true</p> <p>Using JUnit -</p> <p>EXPECT each party's seats to match the number of party seats expected string, otherwise set testPassed to false</p> <p>EXPECT each party's candidates to correspond to the number of party seats and the correct vote order, otherwise set testPassed to false</p>

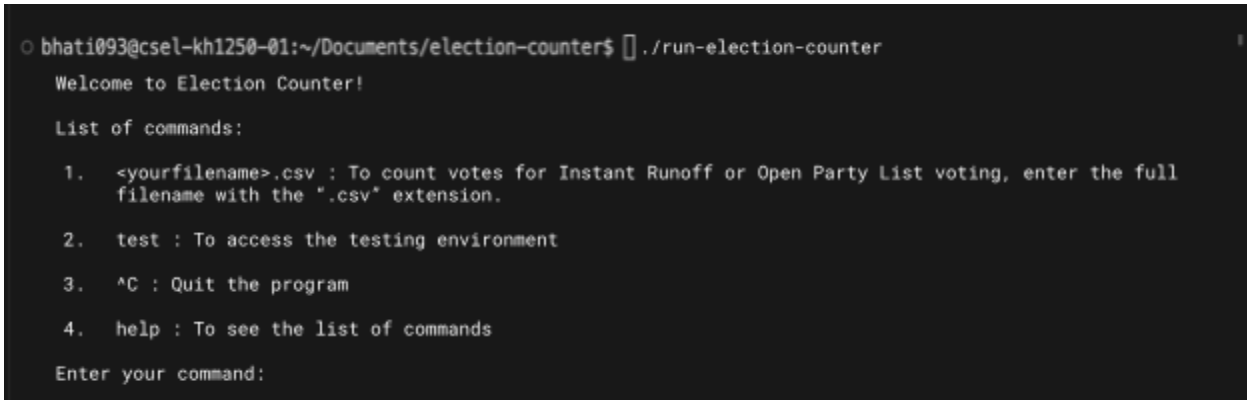
	CALL printTest("Run IR Election", testPassed, expected, electionObj)
--	---

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The Voting system provides users with the ability to count ballots for Instant Runoff and Open Party List voting as well as access to testing environments to ensure that the algorithm is working correctly. This is done through a command line interface. We will provide instructions on how to start the program. Once started, the system will provide additional information on system specific commands, how to input a file to the system, commands to correctly use the testing environment and how to access the help screen. The user will be able to type in those commands when prompted and they will be given feedback on the success or failure of the system recognizing their command. If they successfully input a correct file for vote counting, the system will display the summary of the election, the election winner and the name and location of a corresponding audit file. If they access the testing environment, they will be given instruction on how to input the correct test files and how to interpret the test results.

6.2 Screen Images



```
o bhati093@cse1-kh1250-01:~/Documents/election-counter$ ./run-election-counter
Welcome to Election Counter!

List of commands:

1. <yourfilename>.csv : To count votes for Instant Runoff or Open Party List voting, enter the full
   filename with the ".csv" extension.
2. test : To access the testing environment
3. ^C : Quit the program
4. help : To see the list of commands

Enter your command:
```

Figure 6: Program Start Screen

```
Enter your command: IRElection1.csv
The file was found! Processing Results . . .
Winner found!
-----
                Instant Runoff Election
-----

Winner: Candidate A (Party X) with 60% Majority

-----
Total Votes: 10000
-----
Candidate Breakdown:

Candidate A: 60 % |xxxxxx
Candidate B: 10 % |x
Candidate C: 10 % |x
Candidate D: 20 % |xx

The audit file with details of the election, how votes were assigned can be found in
IRElection1_Audit_10-16-2023-1400.txt in the C:/Documents/election-counter directory.

Enter your command: |
```

Figure 7: Processing an IR election

```

bhati093@cse1-kh1250-01:~/Documents/election-counter$ ./run-election-counter

Welcome to Election Counter!

List of commands:

1. <yourfilename>.csv : To count votes for Instant Runoff or Open Party List voting, enter the full
   filename with the ".csv" extension.

2. test : To access the testing environment

3. ^C : Quit the program

4. help : To see the list of commands

Enter your command: OPLElection1.csv

The file was found! Processing Results . . .

-----
Open Party List Election
-----

Candidates are ordered from most to least popular

No of Votes:      Party A (3 seats)|      Party B (1 seat)      Party C (0 seats)
-----
1. John Doe      6000 |      ABCD      3000
2. Jane Doe      2000 |
3. EFGH          2000 |

The audit file with more details of the election and how votes were assigned can be found in
OPElection1_Audit_10-16-2023-1401.txt in the C:/Documents/election-counter directory.

Enter your command: |

```

Figure 8: Processing an OPL Election

```
Enter your command: test

Welcome to the election-counter testing environment! Please use the following commands to navigate:

1. test file <yourInputFile.csv> : tests if the file is opened correctly and if the header
   information is read correctly.
2. test auditFile <yourInputFile.csv> : tests if the corresponding audit file is created
   appropriately and has read-only permissions. Does NOT test the formatting of the audit file.
3. test IRElection <yourInputFile.csv> : tests if the IR election algorithm runs correctly and finds
   the correct winner and appropriate candidate vote breakdown.
4. test OPLElection <yourInputFile.csv> : tests if the OPL election algorithm runs correctly and
   assigns seats and winning candidates appropriately.
5. test tieBreaker <countOfTies> : runs a tie breaking simulation for the given number of candidates
   for 5000 rounds to see expected value.
6. exit : exit the testing environment
7. test help: display commands for the testing environments

Enter your command: |
```

Figure 9: Accessing the testing Environment

```
Enter your test command: test file IRElection.csv

-----
Testing file-read and file header functions for IRElection.csv!
-----

Enter expected election type acronym (IR/OPL): IR

Enter expected number of candidates: 4

Enter comma separated values of candidates and parties as formatted in the election file
example: Can1 (D), Can 2 (I), Can 4(D), Rosen (D), Kleinberg (R), Chou (I), Royce (L)

Enter number of ballots: 6

Running Tests. . .

-----
TEST FILE - OPEN & HEADER
-----

Expected
-----
File Opened:      True
Election Type:    IR
No of Candidates: 4
Candidate List:   Rosen (D), Kleinberg (R), Chou (I), Royce (L)
No of Ballots:    6

Received
-----
File Opened:      True
Election Type:    IR
No of Candidates: 4
Candidate List:   Rosen (D), Kleinberg (R), Chou (I), Royce (L)
No of Ballots:    6
```

Figure 10: Running a test for file reading


```
Enter your test command: test filx IRElection.csv

Cannot understand command test filx. Please see list of commands.

List of commands:

1. test file <yourInputFile.csv> : tests if the file is opened correctly and if the header
   information is read correctly.
2. test auditFile <yourInputFile.csv> : tests if the corresponding audit file is created
   appropriately and has read-only permissions. Does NOT test the formatting of the audit file.
3. test IRElection <yourInputFile.csv> : tests if the IR election algorithm runs correctly and finds
   the correct winner and appropriate candidate vote breakdown.
4. test OPLElection <yourInputFile.csv> : tests if the OPL election algorithm runs correctly and
   assigns seats and winning candidates appropriately.
5. test tieBreaker <countOfTies> : runs a tie breaking simulation for the given number of candidates
   for 5000 rounds to see expected value.
6. exit : exit the testing environment
7. test help: display commands for the testing environments

Enter your command: test OPLElection OPLElectionFe.csv

Sorry, the file could not be found. Please enter a correct filename.

Enter your command: |
```

Figure 11: Re-entering commands when incorrect spelling is used or file is not found

6.3 Screen Objects and Actions

The above screenshots showcase the Command Line interface for users to access both the testing environment and the election running environment. Figure 1 showcases the start screen where all the commands, their usage and purpose is displayed to the user. Figures 2 and 3 show a sample IR and OPL election being run. Given that user types in the correct command and the appropriate filename, and the system automatically calculates the results and displays it, along with information about where the uniquely named audit file can be found. If the user enters an incorrect command, or a file that cannot be found, they are shown the list of valid commands and prompted to type again (as shown in Figure 6).

7. REQUIREMENTS MATRIX

USE CASE #	Description	Class/Methods
UC_001	Input an election data CSV file into the system	<p>FileProcessor:</p> <ul style="list-style-type: none"> ● readHeader(): reads the initial information from the CSV file to determine the type of election and other meta-data ● readRemainingIR(ElectionType elec) and readRemainingOPL (ElectionType elec): read the rest of the data for IR or OPL elections respectively ● has a mechanism in place that validates file name <p>Controller:</p> <ul style="list-style-type: none"> ● main(): program begins and the Election Official is prompted to enter the filename ● manages the feedback mechanism when an incorrect filename is given and the option to retry is offered
UC_002	Read the input file header to extract relevant election information, such as the type of election (IR or OPL) and the number of candidates and parties	<p>FileProcessor:</p> <ul style="list-style-type: none"> ● readHeader(): reads the file's header, which contains the type of election and relevant metadata (e.g., number of candidates and parties), extracts the type of election (IR or OPL) from the file's first line, and then continues to process the subsequent lines for other relevant information

		<ul style="list-style-type: none"> handles the feedback mechanism when the file cannot be opened <p>ElectionType (or its subclasses IRElection and OPLElection):</p> <ul style="list-style-type: none"> setBallotCount(), setCandidatesCount(), and setCandidateList(): used to set the extracted data into respective ElectionType objects and to store that data for later processing
UC_003	Read the remaining election file after extracting header information to access and extract the ballot data	<p>FileProcessor:</p> <ul style="list-style-type: none"> readRemainingIR(ElectionType elec) and readRemainingOPL(ElectionType elec): depending on whether the election type identified in the header is IR or OPL, the appropriate method is called to read and process the remaining ballot data; these methods will handle the parsing of each line, ensuring the ballot data is extracted correctly based on the format handles the exception case of file access issues <p>ElectionType (and its subclasses IRElection and OPLElection):</p> <ul style="list-style-type: none"> the methods within these classes, particularly any methods that deal with managing and storing ballot data (e.g., createCandidateBallotMap() in IRElection), will be used following the reading of the ballot data <p>Ballot:</p> <ul style="list-style-type: none"> translateBallot(String input): used during the reading process to translate the input ballot

		string into a more structured or processed format suitable for storage and further processing
UC_004	Run an Instant Runoff (IR) election, and in the event of no clear majority, declare the winner based on popularity	<p>IRElection:</p> <ul style="list-style-type: none"> ● <code>runElection()</code>: processes the IR election by tallying the votes, determining if there's a majority winner, and implementing the Instant Runoff mechanism ● <code>tieBreaker()</code>: system falls back to this in the case of no majority winner and determines the winner based on popularity ● <code>displayElectionResults(String electionType, int ballotCount, String winner, <candidate breakdown>, <audit-file-name>)</code>: displays the final results to the public ● <code>writeToAuditFileIR(HashMap<Candidate,List<Ballot>>, candidateBallotMap, String text)</code>: generates an audit file if the program runs uninterrupted and the results need to be logged <p>Candidate:</p> <ul style="list-style-type: none"> ● <code>incrementCandidateVoteCount(int amt)</code>, <code>decrementCandidateVoteCount(int amt)</code>, and <code>getCandidateVoteCount(String candidateName)</code>: used during the process of tallying votes and determining the winner based on popularity in the event of a tie <p>Controller:</p> <ul style="list-style-type: none"> ● exception handling and cleanup taken care of if

		the user stops the program from the CLI
UC_005	Run an Open Party Listing (OPL) election while treating all independent candidates as if they belong to a single party	<p>OPLElection:</p> <ul style="list-style-type: none"> • <code>runElection()</code>: orchestrates the entire process of vote counting, seat assignment, and result calculation specific to the OPL election type • <code>setNumberOfSeats()</code>: used depending on the number of seats available in the OPL election. <p>Candidate:</p> <ul style="list-style-type: none"> • <code>assignCandidateToParty()</code>: crucial when handling independent candidates so that the system can assign all independent candidates to a single "independent" party grouping <p>Party:</p> <ul style="list-style-type: none"> • <code>Party(String party)</code>: the constructor will be useful to create an "independent" party grouping if it doesn't already exist; this is especially important for this use case, where all independent candidates are treated as belonging to a single party • <code>addCandidateToList(Candidate c)</code>: used to add each independent candidate to this single party grouping <p>FileProcessor:</p> <ul style="list-style-type: none"> • <code>readRemainingOPL(ElectionType elec)</code>: parses the input file by checking for the independent status of candidates and groups them together accordingly (the input file will contain information regarding which candidates are

		<p>independent)</p> <p>Controller:</p> <ul style="list-style-type: none"> if the program is halted by the user, the gathered data is discarded, and no audit file is created and there is a mechanism in place to reset or halt processes if the program is interrupted
UC_006	Ballot order is shuffled before votes are counted for OPL elections to prevent unfair bias towards a particular candidate	<p>OPElection:</p> <ul style="list-style-type: none"> runElection(): the order of the ballots is shuffled and votes are counted
UC_007	Test the shuffling of ballots to ensure that the process works correctly	<p>Test:</p> <ul style="list-style-type: none"> testRunIRElection(data, expected) and testRunOPElection(data, expected): run the IR and OPL elections on test data and produces results respectively <p>ElectionType:</p> <ul style="list-style-type: none"> runElection(): has integrated functionality to shuffle ballots before the election procedure
UC_008	Test if the software's Instant Runoff (IR) algorithm functions correctly and accurately processes election results	<p>Test:</p> <ul style="list-style-type: none"> testRunIRElection(data, expected): runs the IR algorithm on test data and produces results <p>IRElection:</p>

	according to the specified rules and logic	<ul style="list-style-type: none"> • <code>runElection()</code>: carries out the IR algorithm and is internally run when <code>testRunIRElection()</code> is triggered to obtain the actual results • <code>displayElectionResults()</code>: used to output the results for comparison by the tester <p>Controller:</p> <ul style="list-style-type: none"> • <code>handleTest()</code>: handles the CLI commands related to testing, including triggering the right test based on provided flags and ensuring the right files are processed and that on abrupt termination, no erroneous or partial results are displayed
UC_009	Tests if the software's Open Party Listing (OPL) algorithm accurately processes election results according to the specified rules and logic to ensure that the IR algorithm functions correctly and produces accurate results	<p>Test:</p> <ul style="list-style-type: none"> • <code>testRunOPLElection(data, expected, shuffle)</code>: tests the OPL election, comparing the results obtained from the software with expected results <p>OPL Election:</p> <ul style="list-style-type: none"> • <code>runElection()</code>: executes the OPL election process and the output is compared with the expected results to determine accuracy • <code>setNumberOfSeats()</code>: used to define the number of seats for the OPL election, if not specified in the test data • <code>displayElectionResults(String electionType, int numberOfSeats, int ballotCount, ArrayList<Candidate> candidateWinners, <candidate breakdown>, auditFileName)</code>: displays the election results and is crucial for

		<p>the tester to review and compare against the expected outcomes</p> <p>Controller:</p> <ul style="list-style-type: none"> • <code>handleTest()</code>: facilitates the testing process and ensures that the program's execution stops gracefully and any gathered data is discarded
UC_010	Break ties in election results by flipping a coin to ensure a fair and impartial resolution	<p>ElectionType:</p> <ul style="list-style-type: none"> • <code>tieBreaker(ArrayList<Object>)</code>: simulates a coin flip and decides the winner impartially when a tie is detected between candidates during the results calculation, given the list of tied candidates or options <p>IElection and OPElection:</p> <ul style="list-style-type: none"> • <code>writeToAuditFile()</code>: logs the tie breaking event and the result of the tie-breaker in the audit file to ensure transparency and provide a record of the tie and its resolution
UC_011	Recreate and validate the process used by the system to determine a winner for the election using an audit file	<p>FileProcessor:</p> <ul style="list-style-type: none"> • <code>createAuditFile()</code>: creates the audit file which comprehensively describes all actions taken by the system; the read-only attribute has a combination of system-specific commands and permissions configurations in place and hence, cannot be modified once it's created <p>IElection and OPElection:</p> <ul style="list-style-type: none"> • <code>writeToAuditFile()</code>: populates the audit files in both election classes by writing down all the

		<p>actions, decisions, and processes the system goes through when determining the election results</p> <p>Controller:</p> <ul style="list-style-type: none"> • <code>main()</code>: triggers the election process, with an output being the winner and the location of the audit file. <p>ElectionType (and its subclasses <code>IRElection</code> and <code>OPElection</code>):</p> <ul style="list-style-type: none"> • <code>displayElectionResults()</code>: outputs the results of either election (including the winner) and provides the location of the audit file for verification
UC_012	The audit file will show the removal of a candidate during the Instant Runoff (IR) voting process for transparency and record-keeping	<p>IRElection:</p> <ul style="list-style-type: none"> • <code>writeToAuditFileIR(HashMap<Candidate,List<Ballot>>, candidateBallotMap, String text)</code>: writes to the audit file during the IR process and records the removal of a candidate • <code>runElection()</code>: determines which candidate to remove based on having the fewest votes and prompts the system to then call <code>writeToAuditFileIR()</code> to log the removal and the associated details in the audit file <p>Candidate:</p> <ul style="list-style-type: none"> • <code>getName()</code>: fetches the name of the candidate who is being removed • <code>getCandidateVoteCount(String candidateName)</code>: fetches the number of votes

		<p>received by the candidate before removal and logs it in the audit file for transparency</p> <p>FileProcessor:</p> <ul style="list-style-type: none"> • createAuditFile(): initially sets up the audit file where all these records would be stored
UC_013	View winners and other election information on the screen after the system has processed the results	<p>IRElection:</p> <ul style="list-style-type: none"> • displayElectionResults(String electionType, int ballotCount, String winner, <candidate breakdown>, <audit-file-name>): displays the election type, ballot count, winner, detailed candidate breakdown, and the name/location of the audit file after processing the IR election results <p>OPElection:</p> <ul style="list-style-type: none"> • displayElectionResults(String electionType, int numberOfSeats, int ballotCount, ArrayList<Candidate> candidateWinners, <candidate breakdown>, auditFileName): displays the election type, number of seats, ballot count, list of winning candidates, detailed candidate breakdown, and the name/location of the audit file after processing the OPL election results
UC_014	The audit file should consist of all the details of the election as soon as a file is inputted into the	<p>FileProcessor:</p> <ul style="list-style-type: none"> • createAuditFile(): generates a blank audit file after a valid ballot data file is entered into the system

	system	<p>IRElection:</p> <ul style="list-style-type: none"> • <code>writeToAuditFile()</code>: writes the details of IR election, including both the input file data and the election results, into the audit file • <code>writeToAuditFileIR(HashMap<Candidate,List<Ballot>>, candidateBallotMap, String text)</code>: writes the details of the IR election into the audit file, including the candidate-ballot map and other related information <p>OPElection:</p> <ul style="list-style-type: none"> • <code>writeToAuditFile()</code>: writes the details of OPL election, including both the input file data and the election results, into the audit file • <code>displayElectionResults(String electionType, int numberOfSeats, int ballotCount,ArrayList<Candidate> candidateWinners, <candidate breakdown>, auditFileName)</code>: displays the results of the OPL election and also updates the audit file with detailed results of the election
--	--------	--

8. APPENDICES

A. Shuffle Feature Revision

- During the initial phase of our requirements specification, we introduced a feature referred to as "shuffle". This was described in use cases UC_007 and UC_008, where the functionality allowed for shuffling of ballots prior to processing election results. The

intention was to introduce an added layer of randomness and impartiality in the voting process.

- The main purpose of our vote counting software system is to ensure accurate and reliable vote counting and introducing a shuffle mechanism did not enhance this core functionality. The introduction of a shuffle feature added unnecessary complexity to the vote processing flow without a clear benefit in terms of voting fairness or system integrity. Furthermore, for auditing purposes, it is essential that the system's operations are transparent and straightforward. Shuffling could introduce questions or doubts about the integrity of the process, even if its purpose was benign.
- Given these considerations, the decision was made to omit the shuffle feature from the final design of the software. Although it appears in the use cases of the requirements specification document and the use cases pertaining to the feature is still present in Section 7 Requirements Matrix, its use has been excluded from the software design document to maintain clarity, simplicity, and focus on the primary objective: accurate vote counting.
- We acknowledge the iterative nature of the software design process and the necessity to continually refine and enhance the system's design based on ongoing evaluations and the exclusion of the shuffle feature is a reflection of this iterative and evaluative process.