Election Tabulation System

Software Design Document

Name (s): Zachary Larsen (lars7566), Amelia Lunning (lunni022), Baanee Singh (sing0931), Josh Subhan (subha013)
Section: 001
Date: (02/24/2025)

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.. 2
2. SYSTEM OVERVIEW.. 3
3. SYSTEM ARCHITECTURE 3
3.1 Architectural Design.. 3
3.2 Decomposition Description.. 4
3.3 Design Rationale.. 7
4. DATA DESIGN 8
4.1 Data Description.. 8
4.2 Data Dictionary.. 8
5. COMPONENT DESIGN.. 10
6. HUMAN INTERFACE DESIGN 15
6.1 Overview of User Interface.. 15
6.2 Screen Images.. 15
6.3 Screen Objects and Actions.. 16
7. REQUIREMENTS MATRIX.. 16

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of the Election Tabulation System. The intended audiences for this document are individuals who are required to know how the system is implemented, such as the development team, testers, project managers, and stakeholders.

1.2 Scope

The Election Tabulation System counts the votes and determines the winner of an election with one or more seats. The system supports two types of voting algorithms: plurality and single transferable vote (STV). The system reads election ballots from a CSV file, calculates results based on the chosen algorithm, displays the results with voting statistics, and generates an audit report detailing ballot allocation. This software is written in Java, and a computer command line interface handles all user interactions. This program aims to reduce human error and the time it takes to calculate an election's results.

1.3 Overview

- Section 1: introduction to the program and its goals.
- Section 2: describes the functionality, context, and design of the program.
- Section 3: outlines the architectural design of the program.
- Section 4: detailed descriptions of the algorithms, data structures, and system components.
- Section 5: examines the software components of the system.
- Section 6: outlines the user interface and design.
- Section 7: cross reference that traces components and data structures to the requirements in the SRS document.

1.4 Reference Material

None.

1.5 Definitions and Acronyms

- CSV: Comma-separated value: A file type where columns of data are separated from one another with a comma.
- STV: Single Transferable Vote: A voting method in which voters submit one ballot with candidates ranked by preference. This method ensures that all winners have won by a majority vote.
- Plurality: The number of votes cast for a candidate who receives more than any other, but does not get an absolute majority

2. System Overview

The Election Tabulation System is designed to count election ballots using two voting algorithms: plurality and STV. Users enter required information including the voting algorithm, the number of seats, and the input ballot file. The program reads ballots from CSV files and calculates the results based on the selected voting method. After processing, the system displays election results and generates an audit report that logs the entire process. It also offers an optional feature to shuffle ballots in STV elections. The system ensures accuracy, transparency, and ease of use, providing a reliable solution for elections with detailed reporting.

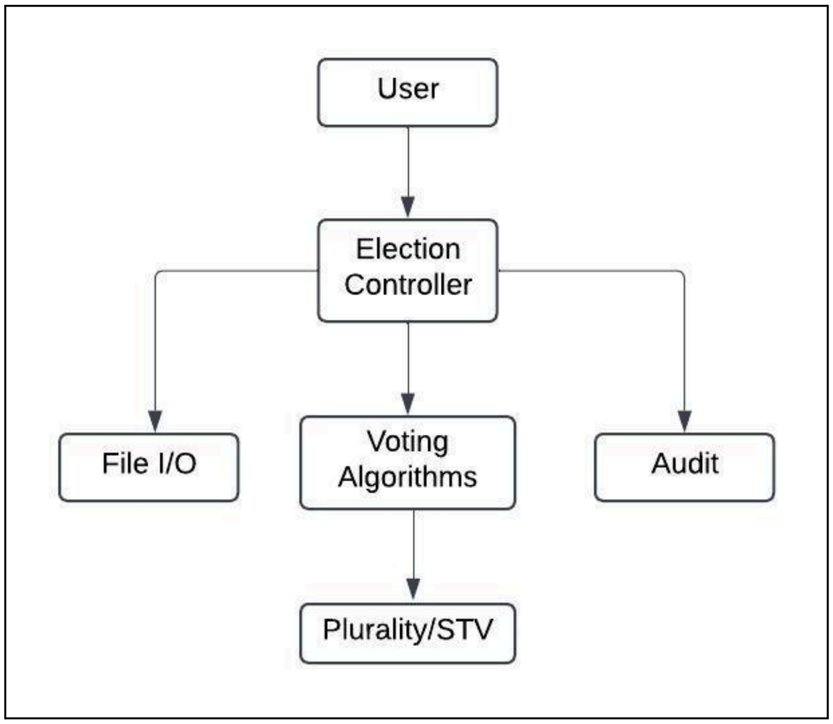
3. System Architecture

3.1 Architectural Design

The Election Tabulation System is divided into the following high-level subsystems with their respective roles and responsibilities:

- User Interface Subsystem
  - Gives visual cues to the user as to when to provide input
  - Shows what the system is doing and the processing steps
  - Displays final election results and statistics
- File Handling Subsystem
  - Allows the user to provide a filename for which ballots are stored
- Election Controller Subsystem
  - Driver for the entire system
  - Calls upon the other subsystems for ballot processing from start to finish
- Voting Algorithm Subsystem
  - Plurality
    - Extracts voter choice from ballots
    - Counts votes for each candidate
  - STV
    - Calculates Droop Quota
    - Shuffles STV ballots
    - Assigns votes
    - Redistributes votes
- Audit Reporting Subsystem
  - Captures all ballot allocation steps
  - Creates a formatted text file with visual indicators of each step

The diagram below illustrates the subsystems and their interconnections:

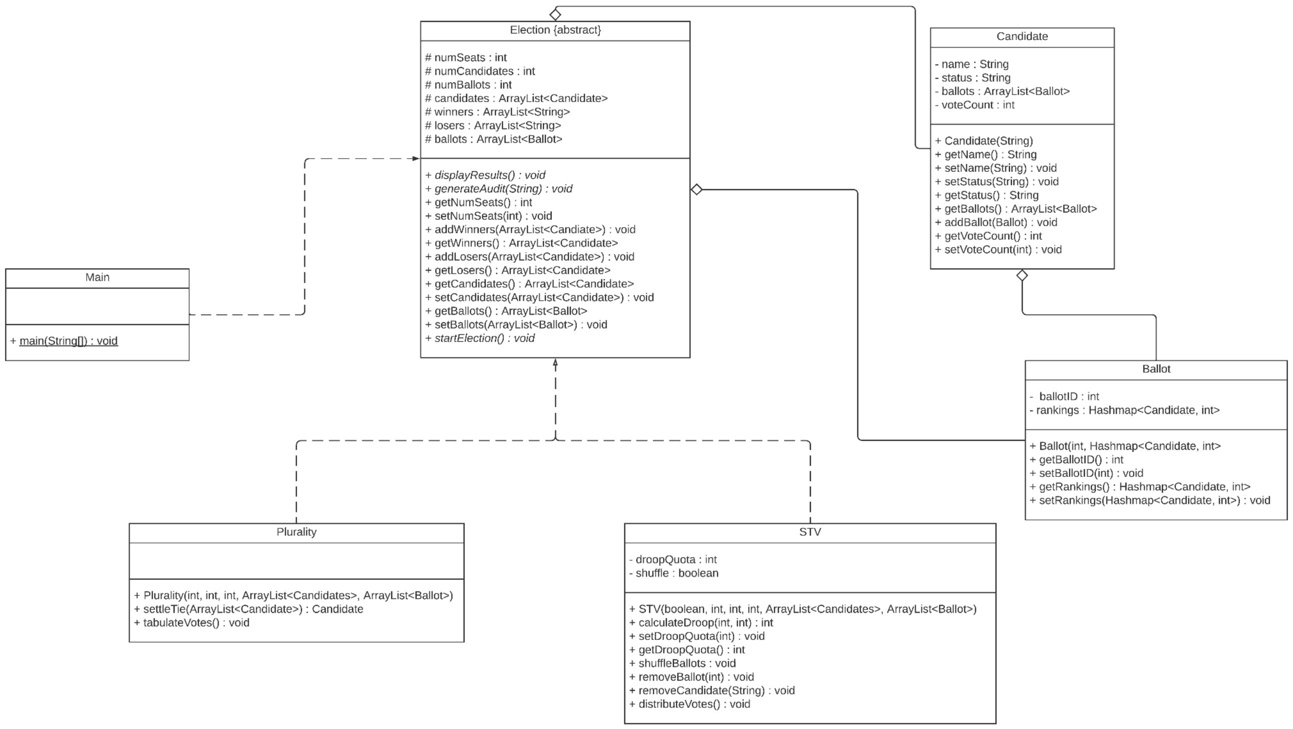


The Election Tabulation System was decomposed in the above manner to keep similar, and highly correlated tasks coupled together. The system requires a driver to keep data moving from one step of the tabulation process to the next. The job of the voting algorithms is to take each ballot and assign it to a candidate based on the rules of the voting method. The main purpose of the file I/O subsystem is to get the ballot file and get the data ready for processing by the rest of the subsystems.

The user starts the system and the election controller subsystem initiates the data flow by prompting the file I/O subsystem to request and process a file from the user. The data from that file moves to the voting algorithms subsystem facilitated by the election controller. The election controller triggers the audit subsystem to generate a document based on the data collected from the voting algorithms.

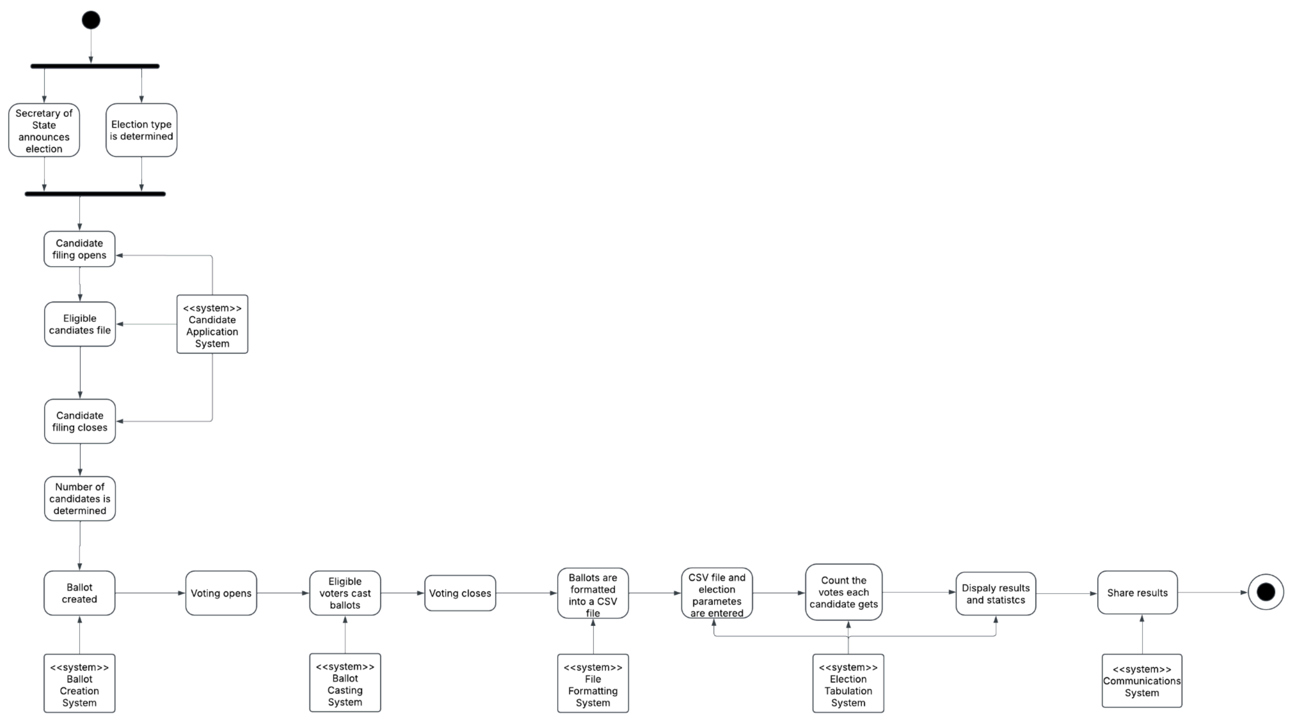
3.2 Decomposition Description

The system follows an object-oriented design with well-defined classes and interactions. A larger, clearer version of all the following diagrams can be found in the SDD folder on GitHub.


Election {abstract}
# numSeats : int
# numCandidates : int
# numBallots : int
# candidates : ArrayList<Candidate>
# winners : ArrayList<String>
# losers : ArrayList<String>
# ballots : ArrayList<Ballot>
+ displayResults() : void
+ generateAudit(String) : void
+ getNumSeats() : int
+ setNumSeats(int) : void
+ addWinners(ArrayList<Candiate>) : void
+ getWinners() : ArrayList<Candidate>
+ addLosers(ArrayList<Candidate>) : void
+ getLosers() : ArrayList<Candidate>
+ getCandidates() : ArrayList<Candidate>
+ setCandidates(ArrayList<Candidate>) : void
+ getBallots() : ArrayList<Ballot>
+ setBallots(ArrayList<Ballot>) : void
+ startElection() : void
Candidate
- name : String
- status : String
- ballots : ArrayList<Ballot>
- voteCount : int
+ Candidate(String)
+ getName() : String
+ setName(String) : void
+ setStatus(String) : void
+ getStatus() : String
+ getBallots() : ArrayList<Ballot>
+ addBallot(Ballot) : void
+ getVoteCount() : int
+ setVoteCount(int) : void
Main
+ main(String[]) : void
Ballot
- ballotID : int
- rankings : Hashmap<Candidate, int>
+ Ballot(int, Hashmap<Candidate, int>
+ getBallotID() : int
+ setBallotID(int) : void
+ getRankings() : Hashmap<Candidate, int>
+ setRankings(Hashmap<Candidate, int>) : void
Plurality
+ Plurality(int, int, int, ArrayList<Candidates>, ArrayList<Ballot>)
+ settleTie(ArrayList<Candidate>) : Candidate
+ tabulateVotes() : void
STV
- droopQuota : int
- shuffle : boolean
+ STV(boolean, int, int, int, ArrayList<Candidates>, ArrayList<Ballot>)
+ calculateDroop(int, int) : int
+ setDroopQuota(int) : void
+ getDroopQuota() : int
+ shuffleBallots : void
+ removeBallot(int) : void
+ removeCandidate(String) : void
+ distributeVotes() : void

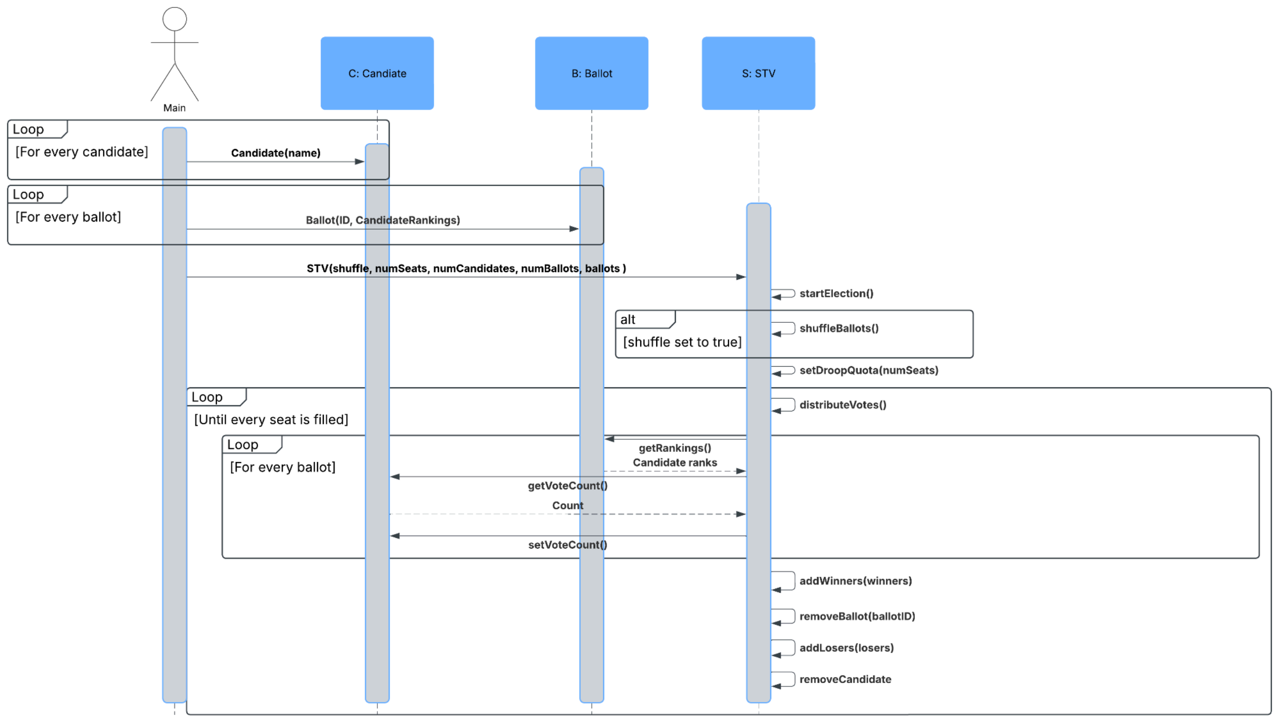
ClassDiagram_Team10.png

This class diagram displays the object-oriented nature of the Election Tabulation System. The diagram demonstrates that we chose to encompass ballots and candidates as their distinct objects that interact with the different election models. The Election abstract class sets up the driving elements of the system that are then utilized by either the Plurality or STV class.


Secretary of State announces election
Election type is determined
Candidate filing opens
Eligible candiates file
<<system>> Candidate Application System
Candidate filing closes
Number of candidates is determined
Ballot created
<<system>> Ballot Creation System
Voting opens
Eligible voters cast ballots
<<system>> Ballot Casting System
Voting closes
Ballots are formatted into a CSV file
<<system>> File Formatting System
CSV file and election parametes are entered
Count the votes each candidate gets
<<system>> Election Tabulation System
Dispaly results and statistcs
Share results
<<system>> Communications System

ActivityDiagram_Team10.png

This activity diagram outlines the entire election process from start to finish, where plurality is the election type of choice. The diagram includes all the systems involved in this process, including our Election Tabulation System, and the steps they are each responsible for handling.



SequenceDiagram_Team10.png

This sequence diagram details the objects, function calls, and respective parameters and return values that are involved in the processing of an STV election. The diagram starts from the point after the user has given the CSV filename and entered the required parameters. The sequence diagram demonstrates how the different objects interact with one another and what data is transferred between them. The process begins with the creation of all candidate and ballot objects and then the STV class takes over by manipulating and organizing those objects based on the order the ballots came in and the candidate rankings they had.

3.3 Design Rationale

The election tabulation system simply runs an election and reports the results of the election as it goes. The chosen architecture for the system is a modular and component-based approach that divides the system into subsystems. This design is motivated by flexibility, scalability, maintainability, and ease of testing. The system is inherently simple, yet complex, dealing with multiple concerns such as user interaction, file handling, vote processing, and result generation. By dividing the overarching system into subsystems (e.g., User Interface, File Handling, Election Controller, Voting Algorithm, Audit Reporting), each subsystem can focus on a specific responsibility. This modularity allows for clearer code organization and easier debugging.

4. DATA DESIGN

4.1 Data Description

The election tabulation system processes ballots stored in CSV files. Candidate data is stored in a list or object, with attributes like name and vote count/ranking. Results are dynamically updated during processing, either by counting votes for plurality or redistributing ballots for STV. The system generates an audit report, tracking each ballot's progression, stored as a text file. Data is handled in memory, with no external databases used, relying on CSV files for input and text files/terminal for output. This structure ensures efficient election processing while maintaining modularity and clear data management.

4.2 Data Dictionary

- **Ballot**
    - Attributes:
        - ballotID: int
        - rankings: HashMap<Candidate, int>
    - Methods:
        - Ballot(int, HashMap<Candidate, int>)
        - getBallotID(): int
        - setBallotID(int): void
        - getRankings(): HashMap<Candidate, int>
        - setRankings(HashMap<Candidate, int>): void
- **Candidate**
    - Attributes:
        - name: String
        - status: String
        - ballots: ArrayList<Ballot>
        - voteCount: int
    - Methods:
        - Candidate(String)
        - getName(): String
        - setName(String): void
        - setStatus(String): void
        - getStatus(): String
        - getBallots(): ArrayList<Ballot>
        - addBallot(Ballot): void
        - getVoteCount(): int
        - setVoteCount(int): void
- **Election (Abstract)**
    - Attributes:
        - numSeats: int
        - numCandidates: int
        - numBallots: int

    - candidates: ArrayList<Candidate>
    - winners: ArrayList<String>
    - losers: ArrayList<String>
    - ballots: ArrayList<Ballot>
  - Methods:
    - displayResults(): void
    - generateAudit(String): void
    - getNumSeats(): int
    - setNumSeats(int): void
    - addWinners(ArrayList<Candidate>): void
    - getWinners(): ArrayList<Candidate>
    - addLosers(ArrayList<Candidate>): void
    - getLosers(): ArrayList<Candidate>
    - getCandidates(): ArrayList<Candidate>
    - setCandidates(ArrayList<Candidate>): void
    - getBallots(): ArrayList<Ballot>
    - setBallots(ArrayList<Ballot>): void
    - startElection(): void
- **Plurality (inherits from Election)**
  - Methods:
    - Plurality(int, int, int, ArrayList<Candidate>, ArrayList<Ballot>)
    - settleTie(ArrayList<Candidate>): Candidate
    - tabulateVotes(): void
- **STV (inherits from Election)**
  - Attributes:
    - droopQuota: int
    - shuffle: boolean
  - Methods:
    - STV(boolean, int, int, ArrayList<Candidate>, ArrayList<Ballot>)
    - calculateDroop(int): int
    - setDroopQuota(int): void
    - getDroopQuota(): int
    - shuffleBallots(): void
    - removeBallot(int): void
    - removeCandidate(String): void
    - distributeVotes(): void
- **Main**
  - Methods:
    - main(String[]): void

5. Component Design

The election system requires these classes to effectively model the process of an election, including ballots, candidates, and different election types. Each class serves a distinct role:
Ballot (every ballot is a hashmap of the candidate ranking)

- Represents an individual voter's choices in an election. It stores a unique ballot ID and a ranking system for candidates, which is crucial for STV elections, where votes are redistributed based on voter preferences. It is included in the diagram as it directly interacts with the Candidate and Election classes, ensuring that each vote is properly recorded and counted. Without this class, the election system would lack a structured way to manage voter input.

```
BEGIN Ballot (ballotID: int, rankings: HashMap<Candidate, int>)
    SET ballotID to given ballotID
    SET rankings to given HashMap of Candidate to int
END

BEGIN getBallotID()
    RETURN ballotID
END getBallotID

BEGIN setBallotID(newBallotID: int)
    SET ballotID to newBallotID
END setBallotID

BEGIN getRankings()
    RETURN rankings
END getRankings

BEGIN setRankings(newRankings: HashMap<Candidate, int>)
    SET rankings to newRankings
END setRankings
```

Candidate

- Represents individuals running in an election. It maintains each candidate's name, status (active or eliminated), vote count, and associated ballots. This class is included in the diagram because it connects with both Ballot and Election, playing a critical role in tracking votes and determining winners and losers. Without a dedicated Candidate class, it would be difficult to store and manage the votes received by each candidate.

```
BEGIN Candidate Constructor (name: String)
    SET name to given name
    SET status to default value ("active")
    INITIALIZE ballots as an empty ArrayList of Ballot
    SET voteCount to 0
END Candidate Constructor

BEGIN getName()
    RETURN name
END getName

BEGIN setName(newName: String)
    SET name to newName
END setName

BEGIN getStatus()
    RETURN status
END getStatus

BEGIN setStatus(newStatus: String)
    SET status to newStatus
END setStatus

BEGIN getBallots()
    RETURN ballots
END getBallots

BEGIN addBallot(newBallot: Ballot)
    ADD newBallot to ballots list
END addBallot

BEGIN getVoteCount()
    RETURN voteCount
END getVoteCount
```

```
BEGIN setVoteCount(newVoteCount: int)
    SET voteCount to newVoteCount
END setVoteCount
```

Election (Abstract)

- An abstract class that serves as the foundation for different election types. It defines core attributes such as the number of seats, candidates, ballots, winners, and losers while also providing common methods like *startElection()*, *displayResults()*, and *generateAudit()*. This class is included in the diagram because it ensures that all elections share a common structure, making it easier to implement multiple election types while maintaining consistency. Without this abstraction, there would be redundant code across different election types, leading to a less maintainable system.

```
BEGIN getNumSeats(): int
        Return numSeats
END

BEGIN setNumSeats(int): void
        Set numSeats
END

BEGIN addWinners(ArrayList<Candidate>): void
        Add Candidate to winners list
END

BEGIN getWinners(): ArrayList<Candidate>
        Return winners list
END

BEGIN addLosers(ArrayList<Candidate>): void
        Add Candidate to losers list
END

BEGIN getLosers(): ArrayList<Candidate>
        Return losers list
END

BEGIN getCandidates(): ArrayList<Candidate>
        Return Candidate list
END

BEGIN setCandidates(ArrayList<Candidate>): void
        Add Candidate to the list
END

BEGIN getBallots(): ArrayList<Ballot>
        Return Ballots list
END

BEGIN setBallots(ArrayList<Ballot>): void
        Add Ballot to Ballots list
END
```

Plurality (Inherits from Election)

- Extends Election and implements Plurality voting, where the candidate with the most votes wins. It includes methods for tabulating votes and handling ties, ensuring that results are correctly determined in a Plurality election. This class is included in the diagram as a subclass of Election because it follows the general election structure but introduces specific behavior unique to Plurality elections. Without this subclass, the system would not be able to handle Plurality-based elections effectively.

```
BEGIN Plurality(numSeats, numBallots, numCanadites, ArrayList<Candidate>, ArrayList<Ballot>)
```

```
        Initialize the constructor for Plurality
        Note: These are protected fields so they can be accessed by the Plurality Class
END

BEGIN settleTie(ArrayList<Candidate>): Candidate
        Get Length of ArrayList
        Get different random ints between 0(inclusive) and Length(exclusive) depending on how many seats to
        fill
        Use those ints to index the ArrayList
        Return the winning Candidates from ArrayList
END

BEGIN tabulateVotes(): void
        Loop through ballots: ArrayList<Ballot>:
                Get Ballot Ranking
                Increment Candidates vote count
        Add correct number of Candidates to winners list
        Settle ties
        Add remaining Candidates to losers list
END

BEGIN dispalyResults() : void
        Print type of election
        Print number of ballots
        Print number of seats
        Print number of candidates
        Print order of winners
        Print order of losers
END

BEGIN generateAudit(String): void
        Create file with specified String name
        Write all information from dispalyResults() to the top of the audit file
        Loop through all Candidates:
                Get Ballots assigned to Candidate
                Loop through Ballots:
                        Get ID of Ballots and write to file
                        Note: This will show the order Ballots came in for specific  Candidates
END

BEGIN startElection(): void
        tabulateVotes()
        displayResults()
        generateAudit()
END
```

STV (Inherits from Election)

- Extends Election to implement STV, a ranked-choice voting system. It introduces attributes such as the Droop quota and a shuffle option, along with methods for redistributing votes, removing candidates, and calculating the quota. This class is included in the diagram as a subclass of Election because STV elections require specialized handling that differs from Plurality elections. Without this class, the system would not be able to support ranked-choice voting, limiting its flexibility.

```
BEGIN STV(shuffle: boolean, int, int, int, ArrayList<Candidate>, ArrayList<Ballot>)
    Initialize state variables
END

BEGIN calculateDroop(int, int): int
    Calculate droop quota
END

BEGIN setDroopQuota(newDroopQuota: int): void
    SET droopQuota to newDroopQuota
END
```

```
BEGIN getDroopQuota(): int
    RETURN droopQuota
END

BEGIN shuffleBallots(): void
    For i=0 to ballots.length - 2 do:
        select random integer j between i and ballots.length-1
        swap ballots[i] and ballot[j]
END


BEGIN removeBallot(ballotID: int): void
    REMOVE ballotId FROM ballots
END

BEGIN removeCandidate(candidateName: String): void
    REMOVE candidateName FROM candidates
END

BEGIN distributeVotes(): void
    While winners.length < numSeats
        Loop through ballots
            Get 1st ranked candidate from ballot
            While candidate not in candidates (candidate elected/eliminated)
                If no next ranked candidate in ballot
                    Remove ballot
                Else get next ranked candidate
            Add ballot to candidate ballots
            Increment candidate vote count
            Remove ballot from ballots
            If candidate vote count = droopQuota
                Append candidate to winners
                Remove candidate from candidates
        If no candidate added to winners during this round
            Find candidate with lowest vote count
            Settle ties
            Add candidate ballots to ballots (to be looped through again and redistributed)
            Add candidate to losers
            Remove candidate from candidates
END

BEGIN dispalyResults() : void
        Print type of election
        Print number of ballots
        Print number of seats
        Print number of candidates
        Print order of winners
        Print order of losers
END

BEGIN generateAudit(filename) : void
        Open a file for writing
        Write data from displayResults() to file
        For every candidate
                For every candidate's ballot
                        Print out the ballotID for the ballot given to that candidate
END

BEGIN startElection() : void
        If the shuffle flag is true:
                shuffleBallots()
        Droop = calculateDroopQuota()
        setDroopQuota(Droop)
        distributeVotes()
        displayResults()
        generateAudit()
END
```

Main

- Serves as the entry point of the program, responsible for initializing elections, processing inputs, and starting the election process. It is included in the diagram separately to distinguish program execution logic from election mechanics. This separation ensures a clear structure, preventing unnecessary dependencies between execution flow and election logic. Without a Main class, there would be no centralized control over the election system, making it difficult to manage and run elections effectively.

```
BEGIN Main
        Prompt user for election type (plurality or STV)
        If STV
                Check for command line shuffle flag
        Prompt user for CSV filename
        Prompt user for number of seats
        Open file
                Retrieve number of candidates
                Retrieve candidate names
        For every candidate
                Create and initialize new Candidate object
                Add to candidate ArrayList
        Retrieve the number of ballots
        For every ballot in the file
                Create and initialize a new Ballot object
                Add to ballots ArrayList
        Instantiate the Election subclass of the user's choice
        Call the class's startElection() method
END
```

6. Human Interface Design

6.1 Overview of User Interface

The user interface of the system is simple. The user will primarily interact with the program through a computer terminal. The program will prompt the user to enter the CSV filename, the election type, the number of seats, and the shuffle on/off. Election results will be displayed on the terminal at the end of the program. This will be the only interaction between the user and the system. Refer to section 6.2 for examples of the system and user interaction via the terminal. Once the program has concluded, the election process and results will be saved to the audit file.

6.2 Screen Images

```
Welcome to the Election Tabulation System!
Please enter the name of your file (must be in .csv):
> voting.txt
File type is not .csv. Please try again.
Please enter the name of your file (must be in .csv):
> voting.csv (Case: not in the same directory)
File not found. Please try again.
Please enter the name of your file (must be in .csv):
> voting.csv
Processing the election file...
File has been processed.
Please enter the election type (STV or Plurality):
> Plurality
Please enter the number of seats to fill:
> [Number of Seats]
Calculating Plurality Votes...
Votes tabulated successfully. Winners and losers have been determined.
Election Summary:
Election Type: [Plurality/STV]
Number of Ballots: [X]
Seats to be Filled: [Y]
Number of Candidates: [Z]
Winners: [List of Winners and their Vote Percentage/Order]
Losers: [List of Losers and their Vote Percentage/Order]
Election summary has been displayed.
Please enter the name of audit file:
> [filename]
Audit has been run...
Results have been saved to the audit file.
Thank you for using the voting algorithm! Goodbye!
```

6.3 Screen Objects and Actions

There are no screen objects used in this system.

7. Requirements Matrix

Use Case ID	Functional Requirements	System Components
UC_01	Filename Prompt	User Interface, File Handling
UC_02	Input Parameters	User Interface, Election Driver

UC_03	Processing CSV File	Election Driver, File Handling
UC_04	Plurality Voting Calculation	Plurality Vote Algorithm
UC_05	STV Voting Calculation	STV Vote Algorithm
UC_06	Audit	Audit Log Generator, File Handling
UC_07	Election Summary Display	Result Display Module
UC_08	System Startup	System Initialization Module