

Team 10

## **Project 1**

### Software Design Document

Zachary Tressel

Isabel Tomb

Ian Luck

Jessica Moore

3/19/20

## TABLE OF CONTENTS

<b>1.</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Purpose	1
1.2	Scope	1
1.3	Overview	1
1.4	Reference Material	1
1.5	Definitions and Acronyms	1
<b>2.</b>	<b>System Overview</b>	<b>2</b>
2.1	User Classes and Characteristics	2
2.2	Design and Implementation Constraints	2
2.3	Assumptions and Dependencies	2
<b>3.</b>	<b>SYSTEM ARCHITECTURE</b>	<b>2</b>
3.1	Architectural Design	3
3.2	Decomposition Description	3
3.3	Design Rationale	6
<b>4.</b>	<b>DATA DESIGN</b>	<b>7</b>
4.1	Data Description	7
4.2	Data Dictionary	7

<b>5.</b>	<b>COMPONENT DESIGN</b>	<b>10</b>
5.1	Election Interface and Child Classes	10
5.1.1	runElection	
5.1.1.1	PluralityElection Implementation	
5.1.1.2	STVElection Implementation	
5.1.2	displayStats	
5.1.2.1	PluralityElection Implementation	
5.1.2.2	STVElection Implementation	
5.1.3	generateBallots	
5.1.3.1	PluralityElection Implementation	
5.1.3.2	STVElection Implementation	
5.2	Ballot Interface and Child Classes	12
5.2.1	getChoice	
5.1.1.1	PluralityBallot Implementation	
5.1.1.2	STVBallot Implementation	
5.3	Candidate Class	13
5.4	BallotGenerator Class	13
5.4.1	createBallots	
5.4.2	shuffle	
5.5	Report Class	14
5.5.1	createLog	
5.5.2	fileError	
<b>6.</b>	<b>HUMAN INTERFACE DESIGN</b>	<b>15</b>
6.1	Overview of User Interface	15
6.2	Screen Images	15

6.3	Screen Objects and Actions	18
<b>7.</b>	<b>REQUIREMENTS MATRIX</b>	<b>18</b>
<b>8.</b>	<b>APPENDICES</b>	<b>20</b>

# 1. INTRODUCTION

## 1.1 Purpose

This software design document describes the architecture and system design of a voting system which can handle two different types of elections. The intended audience is developers and anyone else who wants to understand how this software is designed.

## 1.2 Scope

The software described in this document is for a voting system. The software will allow the user to run two different types of voting systems: a plurality system and a single transferable voting (STV) system using a droop quota. The user will input how many seats there are to fill and the algorithm to use at the beginning of the program. The user can also run a test file and turn the shuffle option off. After the software has been run, the results of the election will be displayed on the screen.

## 1.3 Overview

This document is divided into sections, each about a different aspect of the design of this software system. Section two provides a system overview, which includes user classes and implementation constraints. Section three describes the system architecture, including the architectural design and design rationale. Sections four, five, and six describe data, component, and human interface design respectively. Finally, section seven describes the requirements matrix while section eight is an appendix. This document can be read in any order, including start to finish.

## 1.4 Reference Material

Team 10's GitHub repository:

<https://github.umn.edu/umn-csci-5801-002-s20/repo-Team10>

Provided SDD Template

<https://canvas.umn.edu/courses/158173/files/11340043/download?wrap=1>

## 1.5 Definitions and Acronyms

- **STV** (Acronym): Single Transferable Voting; a ranked choice system of voting that utilizes Droop quota to declare candidates winners.

- **Droop Quota:** The smallest number that guarantees that no more candidates can reach the quota than the number of seats available to be filled, calculated by the following equation:

$$\left\lceil \left( \frac{\text{numberOfBallots}}{\text{numberOfSeats}} + 1 \right) \right\rceil + 1$$

## 2. SYSTEM OVERVIEW

This system counts and organizes votes to determine the winner(s) of a small scale election. It has a simple, text based user interface. The system has the ability to run two different types of elections, plurality or STV. It eliminates the need for anyone to manually count and organize completed ballots. This software was designed to be a new self-contained product.

### 2.1 User Classes and Characteristics

The User class that we anticipate using this voting system would primarily be election officials and the people responsible for monitoring and ensuring valid voting results are captured. The election officials are the most important user class for this system because they are responsible for capturing voting results and this system will provide those results. The system must be multifaceted and be capable of satisfying requirements directly pertaining to the user class such as performing the STV method or the plurality voting method depending on what is required by the election officials. Developers/testers may also use this software but will not have any special permissions or a different class.

### 2.2 Design and Implementation Constraints

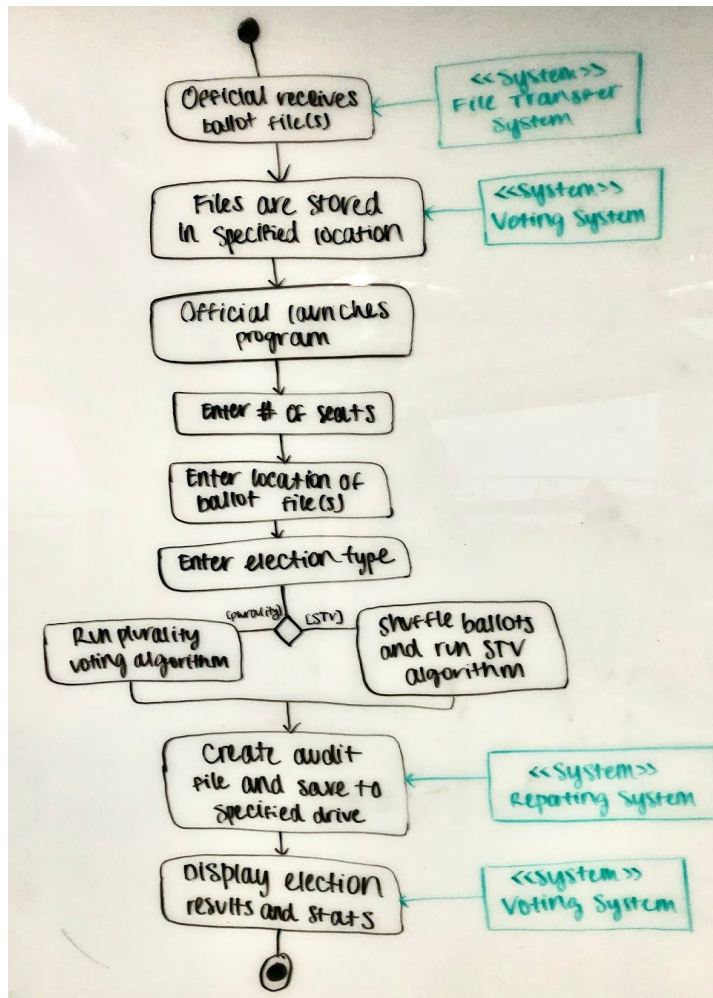
This software will be coded in Java and the user interface will be written in English. The software and relevant hardware must be able to process up to 100,000 ballots in under 5 minutes.

### 2.3 Assumptions and Dependencies

It is assumed that the users have been correctly trained on how to use the software. The software will also assume that there are no mistakes in the input files and the files are Windows CSV files. We assume that users will provide valid input for the number of seats, voting algorithm type, and location of files. We also assume that the security of the voting is ensured at the voting precinct itself. Additionally, the software assumes that the user will only need to turn ballot shuffling off for the STV algorithm for testing purposes, so it is only permitted as a special input argument.

## 3. SYSTEM ARCHITECTURE

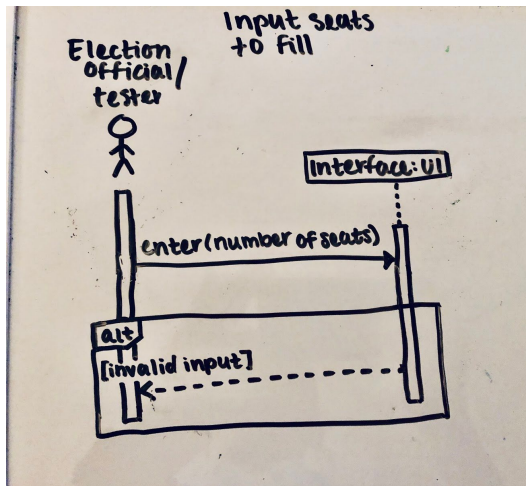
### 3.1 Architectural Design



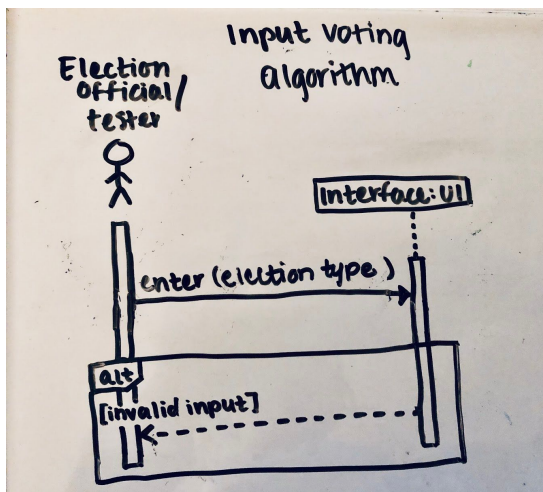
The overall system of this software is the Voting System. This system supports interacting with the user to gather required information about the election and running the election algorithm itself. Within the overall Voting System, we interact with the File Transfer System and the Reporting System. The File Transfer System is an external system that handles the transfer of ballot files from the voting locations to the election official, who then inputs the files into the Voting System, as described in the diagram above. The Reporting System handles the creation and formatting of a log file to be used for auditing purposes. It gathers information about how the ballots were distributed amongst candidates and calculates permanent statistics to be included in the log file, as shown at the end of the diagram above.

### 3.2 Decomposition Description

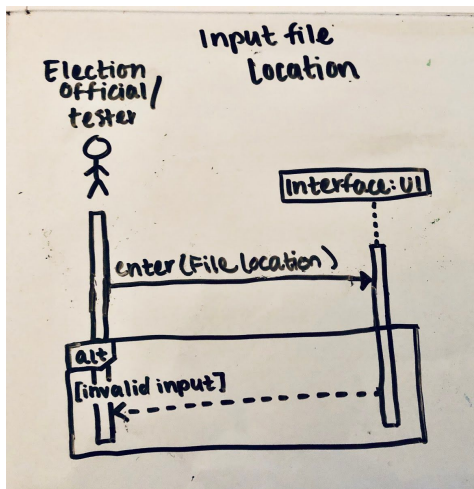
Use case 1: User inputs how many seats there are to fill



Use case 2: User inputs the algorithm to use

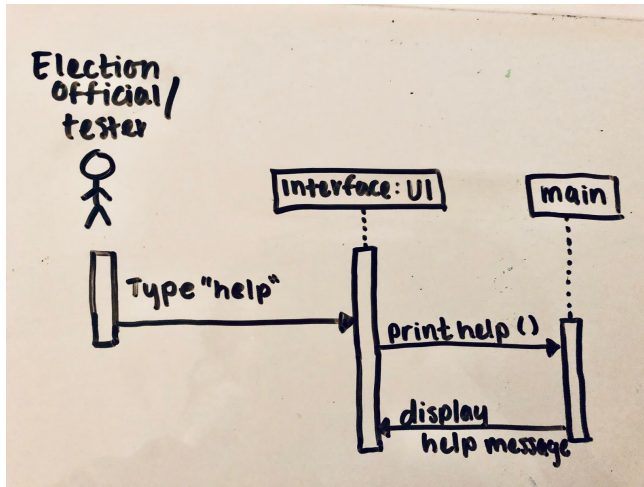


Use case 3: User inputs location of files

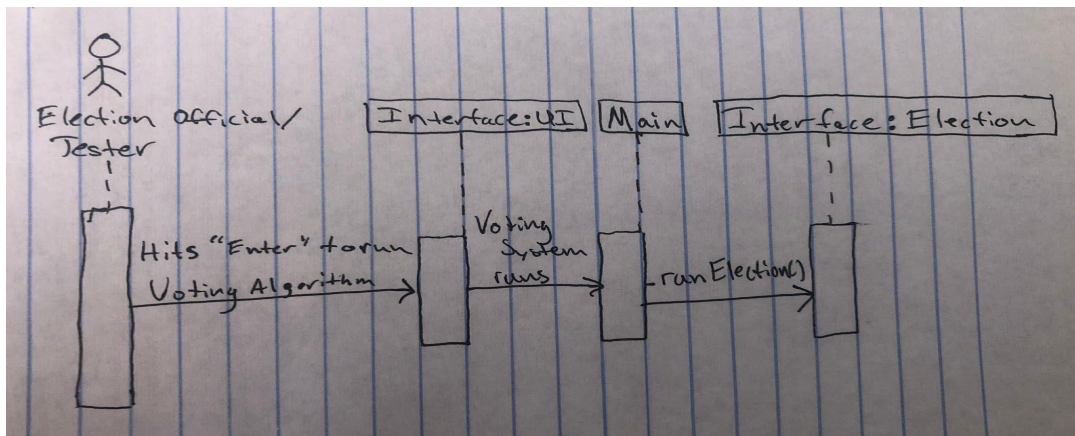




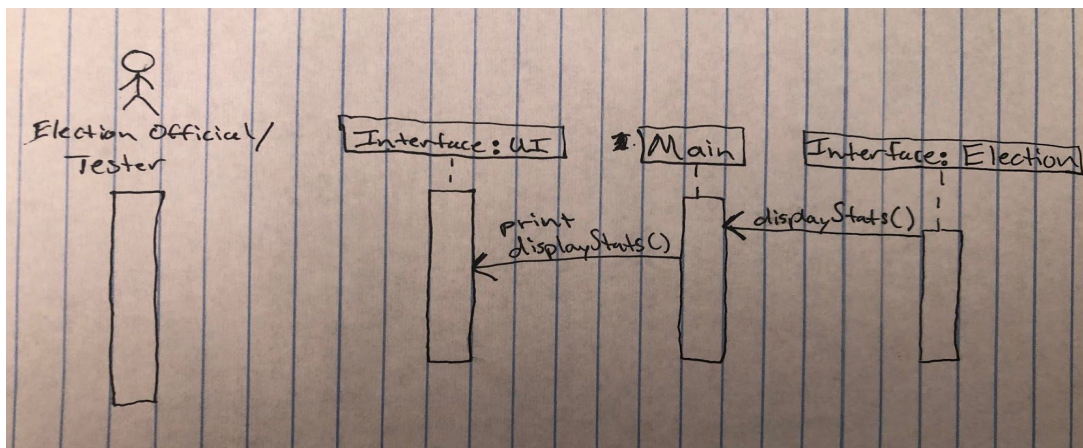
Use case 4: User opens help menu



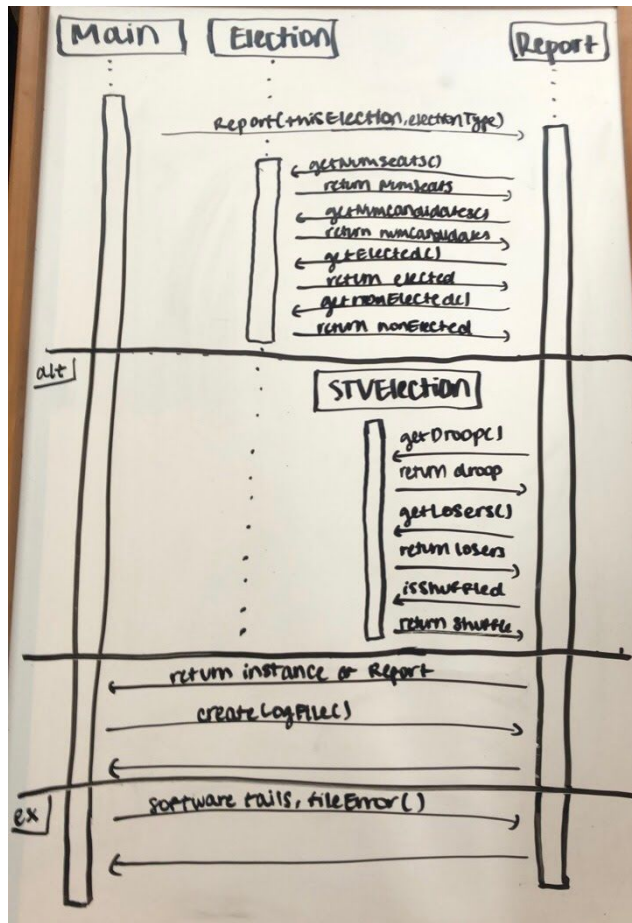
Use case 5: User runs the election



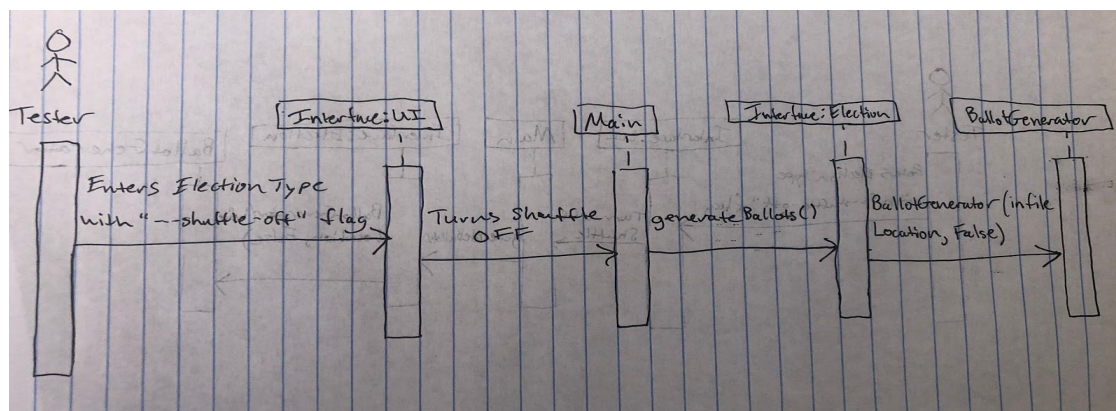
Use case 6: User views election results



Use case 7: User creates report



Use case 8: User (tester) turns the shuffle option on/off



### 3.3 Design Rationale

First we broke the system down into the subsystems discussed in section 3.1. Initially we considered including a balloting system as one of the subsystems. However because the user would be providing the ballot files and giving their location it was decided that a balloting system was not necessary to include at this point because we would have no interaction with it.

In the future if this software were to be developed further the inclusion of a balloting system may be necessary.

We started designing the system by choosing the main classes that would make up the system. We began with the following classes: plurality election, STV election, candidate, plurality ballot, STV ballot, and generator. As we began to create diagrams and decide which methods would be needed in each class it became apparent that there was a significant amount of overlap between the plurality election class and the STV election class. To minimize duplication of code it was decided that an interface called election, with two child classes, plurality election and STV election would be used instead. The same realization was made regarding the two ballot classes so a ballot interface was made as well.

## 4. DATA DESIGN

### 4.1 Data Description

For both plurality and STV, two hashmaps will be used. Both will be of type <Candidate, LinkedList(Ballot)>. The “nonElected” hashmap will contain all candidates and will be used to store all candidates that haven’t won or lost yet. The “elected” hashmap will contain all candidates that have been deemed elected and in the order that they were elected. For audit purposes, each candidate will be connected to a list of ballots that voted for them. In STV elections, an array list of type Candidates will also be kept track of. This will keep track of all candidates that are eliminated from the “nonElected” hashmap and the order that they were added.

### 4.2 Data Dictionary

Object	Attributes	Methods
Ballot <<Interface>>	<b>candidates</b> - candidate array <b>votes</b> - integer array	<b>getChoice()</b> - getter method which returns the chosen candidate when called on a ballot
BallotGenerator	<b>fileLocation</b> - string <b>shuffle</b> - boolean	<b>BallotGenerator(string inFileLocation, boolean shuffle)</b> - constructor which takes in a string indicating the location of the ballot file(s) and a boolean value which serves as a flag indicating if ballot shuffling should be done or not, returns a new ballotGenerator object <b>createBallots(HashMap&lt;Candidate, LinkedList(Ballot)&gt; candidates)</b> -

		<p>parses files at given location to fill the hashmap with candidates given in file and create instances of ballot objects</p> <p><b>shuffle(ArrayList&lt;Ballot&gt; ballots)</b> - shuffles the ballots into a random order</p>
Candidate	<p><b>index</b> - integer</p> <p><b>name</b> - string</p> <p><b>numVotes</b> - integer</p>	<p><b>Candidate(int index, string name)</b> - constructor which returns a new candidate object with index being the candidate's index in the ballot file and name being their name</p> <p><b>getVotes()</b> - getter method which returns the value stored in the numVotes attribute associated with the candidate object</p> <p><b>getName()</b> - getter method which returns the string stored in the name attribute associated with the candidate</p> <p><b>addBallot()</b> - method which increments the numVotes attribute by one for the given candidate object</p>
Election <<Interface>>	<p><b>elected</b> - hashmap containing candidate objects each with a linked list of the associated ballots</p> <p><b>nonElected</b> - hashmap containing candidate objects each with a linked list of the associated ballots</p> <p><b>votes</b> - ArrayList of ballots</p> <p><b>numCandidates</b> - integer</p> <p><b>numSeats</b> - integer</p> <p><b>ballotFileLocation</b> - String</p>	<p><b>runElection()</b> - method to begin running the algorithm for the selected election type</p> <p><b>displayStats()</b> - outputs the elected, nonElected, votes, numCandidates and numSeats attributes associated with the election object</p> <p><b>generateBallots()</b> - method used to call the BallotGenerator class</p> <p><b>getBallots()</b> - method used to return the votes ArrayList&lt;Ballot&gt; attribute</p> <p><b>getNumSeats()</b> - getter method which returns the value stored in the numSeats attribute of the election object</p> <p><b>getNumCandidates()</b> - getter method that returns the number of candidates</p> <p><b>getElected()</b> - getter method that returns the hashmap of elected candidates and their assigned ballots</p> <p><b>getNonElected()</b> - getter method that returns the hashmap of unelected candidates and their assigned ballots</p>

PluralityBallot - inherits from the ballot interface		<b>PluralityBallot(int[] votes, Candidate[] candidates)</b> - constructor which takes in an integer array and a candidate array, and returns a new plurality ballot object
PluralityElection - inherits from the election interface		<b>PluralityElection(int numSeats, string ballotFileLocation)</b> - constructor which takes in an integer numSeats, the number of seats to be filled in the election and a string which is the ballot file location, returns a new plurality election object
Report	<b>electionType</b> - string <b>seats</b> - integer <b>numCandidates</b> - integer <b>elected</b> - array of candidate objects <b>losers</b> - array of candidate objects <b>logFileName</b> - String	<b>Report(election currentElection)</b> - constructor which takes in the election object for the election being currently run and creates a new report object <b>createLog()</b> - method to create, open and write results to the log file, which is the auditable file created during each election, detailing the process <b>fileError()</b> - records an error message at the bottom of the log file if software fails
STVBallot - inherits from the ballot interface	<b>currNum</b> - integer <b>ballotId</b> - integer	<b>STVBallot(int[] votes, Candidate[] candidates)</b> - constructor which takes in an integer array of votes, a Candidate array of candidates, and returns a new STVBallot object <b>setBallotId()</b> - setter method that sets the ballotId attribute
STVElection - inherits from the election interface	<b>losers</b> - array list of candidates <b>droop</b> - integer <b>shuffle</b> - boolean	<b>STVElection(int numSeats, string ballotFileLocation, boolean shuffle)</b> - constructor which takes in an integer for the number of seats in the election, a string with the ballot file location and a boolean value indicating if the ballots should be shuffled or not, returns a new STVElection object <b>getLosers()</b> - getter method that returns the array of losing candidates <b>getDroop()</b> - getter method that returns

		the value of Droop for the election <b>isShuffled()</b> - getter method that returns whether or not the ballots should be shuffled
--	--	---

## 5. COMPONENT DESIGN

### 5.1 Election Interface & Child Classes

The Election interface implements the voting method in question. To do so, it creates ballots given by the file inputted by the election official, reads the ballots based on the chosen voting method, and displays the results of the election. It is implemented by PluralityElection and STVElection, which determines the winning candidate(s) of the election by their respective algorithms.

#### 5.1.1 runElection

This function reads the ballots by calling the generateBallots function and sorts through them based on the chosen voting method, and determines the winner(s) of the election. It then calls displayStats to show the winner and statistics

##### 5.1.1.1 PluralityElection implementation

```
runElection()
    iterate through all ballots
        increment the vote total for the respective candidate
    while numSeats > size of elected
        pick candidate with highest vote total (if tie, break randomly)
        remove them from nonElected and add them to elected
```

##### 5.1.1.2 STVElection implementation

```
runElection()
    droop is calculated
    iterate through all ballots
        while ballot.getChoice is not in nonElected
            increment currNum
            increment that Candidate's vote total
            add the ballot to that Candidate's linked list
        If the candidate has reached droop
            remove from nonElected and add to Elected
        if numSeats == size of elected
```

```

        break

while nonElected isn't empty and numSeats > size of elected
    remove Candidate in nonElected with lowest vote total
    add that candidate to losers
    take that candidate's list of ballots and add them to queue of votes
    iterate through all newly added ballots
        while ballot.getChoice is not in nonElected
            increment currNum
        increment that Candidate's vote total
        add the ballot to that Candidate's linked list
    If the candidate has reached droop
        remove from nonElected and add to Elected
        if numSeats == size of elected
            break

ind = losers.size - 1
while numSeats < size of elected
    add loser at index ind to elected
    remove that candidate from loser
    decrement ind

```

### 5.1.2 displayStats

This function is a helper function for runElection. It creates

#### 5.1.1.1 PluralityElection implementation

```

displayStats()
    Println("Election Type: Plurality")
    Println("Seats:", getNumSeats)
    Println("Number of Candidates:", getNumCandidates)
    Println("The order in which the candidates are listed, is the order that they
were elected.")
    print("Elected:")
    int total_votes = getBallots().size()
    for(Map.Entry<Candidate, LinkedList(Ballot)> getElected().entrySet()){
        print(entry.getKey().getName())
        print(" | % Votes: ", (entry.getKey().getVotes()/total_votes)*100)
        println()
    }
    println()
    print("Non-Elected:")
    for(Map.Entry<Candidate, LinkedList(Ballot)> entry : getLosers().entrySet()){
        print(entry.getKey().getName())
    }

```

```
    print(", ")
}
```

### 5.1.1.2 STVElection implementation

```
displayStats()
    println("Election Type: STV\n")
    println("Seats:", getNumSeats)
    println("Number of Candidates:", getNumCandidates)
    println("The order in which the candidates are listed, is the order that they
were elected.")
    print("Elected:")
    for(Map.Entry<Candidate, LinkedList(Ballot)> getElected().entrySet()){
        print(entry.getKey().getName())
        print(", ")
    }
    println()
    print("Non-Elected:")
    for(Map.Entry<Candidate, LinkedList(Ballot)> entry : getLosers().entrySet()){
        print(entry.getKey().getName())
        print(", ")
    }
```

### 5.1.3 generateBallots

This function is a helper function for runElection. It creates an instance of the BallotGenerator class to parse the candidates and ballots. The implementations of the Election interface differ slightly between its children.

#### 5.1.3.1 PluralityElection implementation

```
generateBallots()
    bg = new BallotGenerator(ballotFileLocation, false, "Plurality")
    votes = bg.createBallots(nonElected)
```

#### 5.1.3.2. STVElection implementation

```
generateBallots()
    bg = new BallotGenerator(ballotFileLocation, shuffle, "STV")
    votes = bg.createBallots(nonElected)
```

## 5.2 Ballot Interface & Child Classes

The Ballot interface contains the votes cast by a voter. It is implemented by PluralityBallot and STVBallot, which read votes cast by the voter by their respective voting methods.

### 5.2.1 getChoice



This function returns the vote cast by a voter.

#### *5.2.1.1 PluralityBallot Implementation*

```
getChoice()
  For i = 0 to votes.length
    if votes[i] == 1
      return candidates[i]
```

#### *5.2.1.2 STVBallot Implementation*

```
getChoice()
  For i = 0 to votes.length
    if votes[i] == currNum
      return candidates[i]
```

### **5.3 Candidate Class**

The Candidate class represents a candidate that is running for office. It contains the index that is associated with it in the original ballot file so that the election algorithm can associate the choice made by a ballot with the correct candidate.

### **5.4 BallotGenerator Class**

The BallotGenerator class is responsible for parsing through the given file(s) in order to create an ArrayList of ballots for the election algorithm to sort through

#### **5.4.1 createBallots**

This function parses the file(s) given by the election official that contain the candidates and the ballots cast by voters. Its parameter, candidates, is the empty HashMap of unelected candidates to fill with candidates. It returns an ArrayList of instances of Ballot.

```
createBallots(HashMap<Candidate, LinkedList<Ballot>> candidates)
  delim = ","
  fileLocations[] = fileLocation.split(delim)
  ballots = new ArrayList<Ballot>()

  for (String file : fileLocations)
    br = new BufferedReader(new FileReader(fileLocation))
    currLine = ""

    while (currline = br.readLine()) != null
      if candidates == null
        String[] candList = currLine.split(delim)
        candidates = new HashMap<Candidate, LinkedList<Ballot>>()
        for (i = 0 to candList.length)
          candidates.put(new Candidate(candList[i], i), null)
```

```

else
    int[] votes = castToIntegers(currLine.split(delim))
    if electionType == "Plurality"
        ballots.add(new PluralityBallot(votes))
    else
        ballots.add(new STVBallot(votes))

if shuffle
    shuffle(ballots)
for i = 0 to ballots.size()
    ballots.get(i).setBallotId(i)

return ballots

```

### 5.4.2 shuffle

The shuffle function shuffles the array of ballots into a random order.

```

shuffle(ballots ArrayList<Ballot>)
    new randomNumberGenerator
    for j = 0 to j = 1000
        for i = 0 to i = size of ballots ArrayList<Ballot>
            generate random number
            swap values at ballots.get(i) and ballots.get(random number)

```

## 5.5 Report Class

### 5.5.1 createLog

This function creates the log file that contains the order the ballots were read by the algorithm, how the winner(s) and loser(s) were determined, and pertinent election statistics.

```

CreateLog()

create logfile "logfile_name"

open(logfile_name)

write to file("Elected:")

for each candidate in elected

    write to file candidate name

    for each ballot in linkedlist of ballots associated with candidate

        write to file (ballot information)

```

```
write to file("\n")

write to file("Non-Elected:")

for each candidate in losers

    write to file candidate name

    for each ballot in linkedlist of ballots associated with candidate

        write to file (ballot information)

    write to file("\n")

write to file("Election Type:" Election Type)

write to file("Seats:", getNumSeats)

write to file("Number of Candidates:", getNumCandidates)

close(logfile_name)
```

### 5.5.2 fileError

This function allows any error messages produced by the software to be recorded at the end of the log file.

```
fileError()
    FileWriter write = new FileWriter(logFileName)
    write to logFile "An error occurred in the election process"
    write.close()
```

## 6. HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

The system will utilize a text based user interface. Text prompts will ask the user to input the necessary information such as, election type, number of seats, and the location of the ballot files. The user will have the option to view a help screen by typing the word "help", the help screen will display instructions for running the program. After an election has been run the election results will be displayed to the user through the user interface.

### 6.2 Screen Images

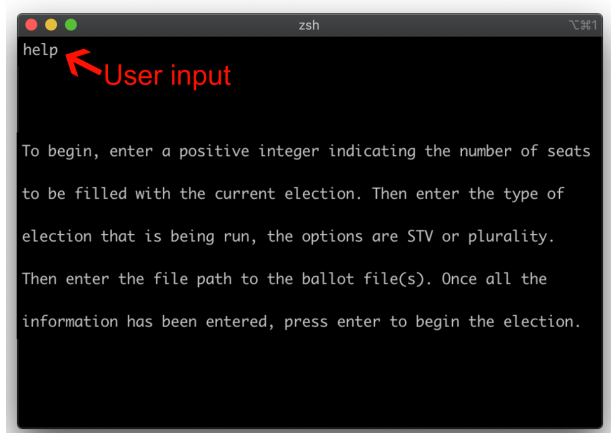


Figure 1: Help screen, displayed after user enters "help"

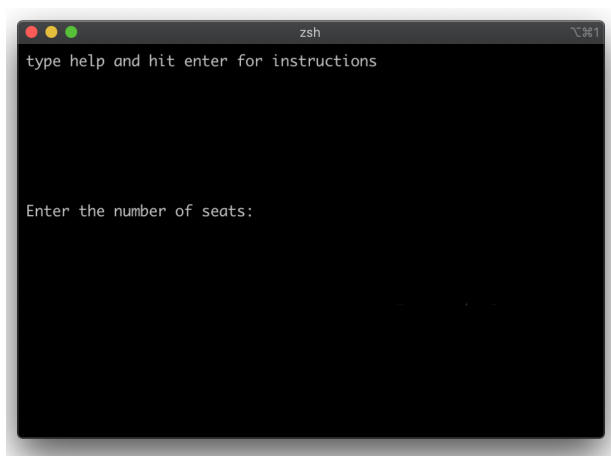


Figure 2: Main screen when the software is initially run

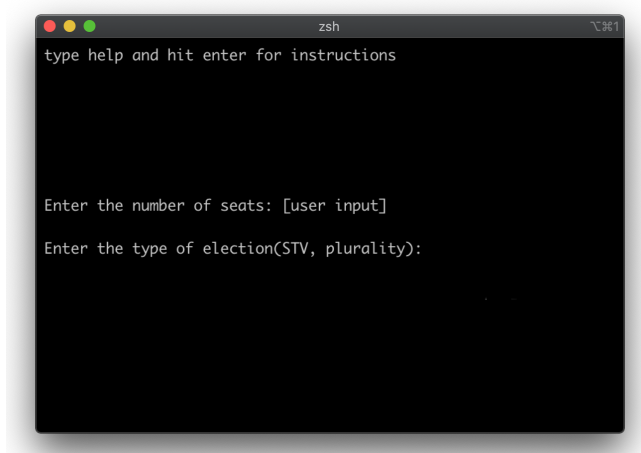


Figure 3: Display after user has entered the number of seats in the election

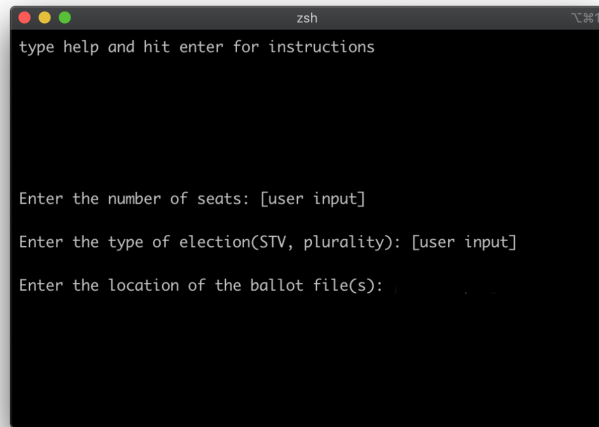


Figure 4: Display after the user has entered the election type

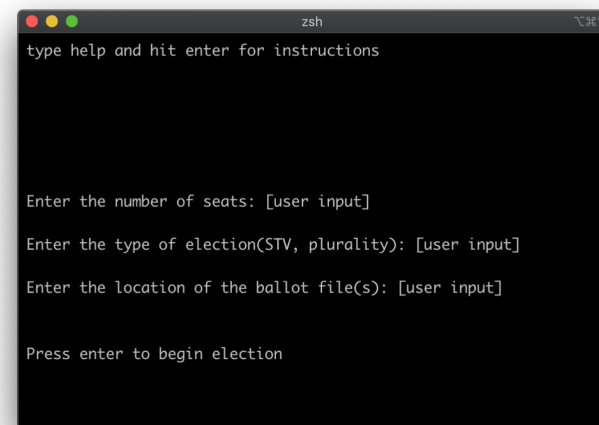


Figure 5: Display after user has entered the file location

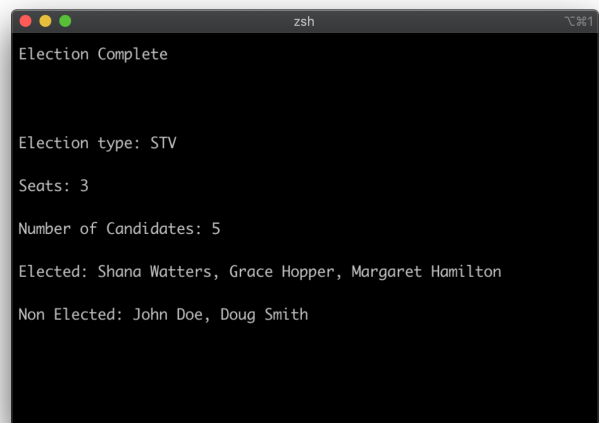
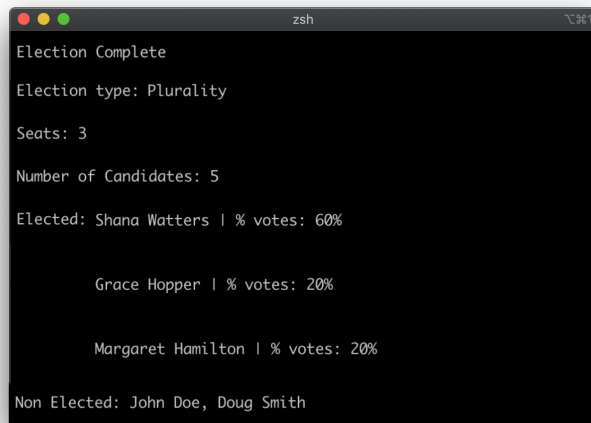


Figure 6: Display after an STV election has completed



```

Election Complete
Election type: Plurality
Seats: 3
Number of Candidates: 5
Elected: Shana Watters | % votes: 60%
Grace Hopper | % votes: 20%
Margaret Hamilton | % votes: 20%
Non Elected: John Doe, Doug Smith

```

Figure 7: Display after plurality election has completed

### 6.3 Screen Objects and Actions

Because we are using a text based interface there are not many screen objects to be discussed. One of the screen objects is the help option. If the user types "help" and presses enter a help message with instructions on how to run the program will be displayed, as seen in Figure 1. The onscreen prompts ask the user to enter the necessary information for running an election. These prompts can be seen in Figures 2, 3 and 4. The last screen object would be the results display. At the end of a successful election the election results are printed to the screen, as seen in Figure 6.

## 7. REQUIREMENTS MATRIX

Below is the requirements matrix indicating which methods and classes take care of the functional requirements that were outlined in the SRS document. The Req ID can be used to refer to the original use cases as outlined in the SRS.

Req ID	Description	Test Strategy	Comments	Location in code
UC1	Input Seats to Fill	Unit test	None at this point	The Election interface has a variable called numSeats. This variable is set by the STV and plurality election constructor methods.
UC2	Input Voting Algorithm	Unit test	None at this point	The Election interface is implemented by the

				PluralityElection and STVElection classes, and the user's input determines which implementation is used in Main.
UC3	Input File Location	Unit test	None at this point	The Election interface has a variable called ballotFileLocation. This variable is set by the STV and plurality election constructors and passed to the BallotGenerator class to create ballots.
UC4	Help Menu	Unit test	None at this point	The main class will handle displaying the help menu. When a user enters the word help into the interface text will appear instructing the user on how to run an election.
UC5	Run Election	Unit test	None at this point	The election interface contains a method called runElection which starts the algorithm for the election type selected by the user (STV or Plurality)
UC6	View Election Results	Unit test	None at this point	The election interface contains a method called displayStats which prints the information related to the election results to the screen
UC7	Create Log File	Unit test	None at this point	The report class has a method called createLog, this method creates and opens the log file that will be written to in order to have an auditable log file showing the steps of the election
UC8	Turn Shuffle Option Off	Unit test	None at this point	The STVElection class has a boolean attribute used to track if shuffle has been turned off. By default the value is set to true and the shuffle method will be run on the ballot array list however a command line argument

				--shuffle-off will set the shuffle attribute to false and the shuffle method will not be run. This is for testing purposes only.
--	--	--	--	--

## 8. APPENDICES

N/A