

Software Design Document (SDD) Template

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main components and providing a general idea of a project definition report. For your own information, please refer to [IEEE Std 10161998](#)¹ for the full IEEE Recommended Practice for Software Design Descriptions.

¹ <http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf>

Team 21

Voting System

Software Design Document

Name (s): Conor Brown (brow4339), Jack Soderwall (soder365), Joe Cassidy (cassi083), Sean Carter (carte899)

Lab Section: N/A

Workstation: N/A

Date: 2/23/2021

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose	3
1.2 Scope	3
1.3 Overview	3
1.4 Reference Material	3
1.5 Definitions and Acronyms	3

2. SYSTEM OVERVIEW

4

3. SYSTEM ARCHITECTURE

4

3.1 Architectural Design	4
3.2 Decomposition Description	6
3.3 Design Rationale	7

4. DATA DESIGN

8

4.1 Data Description	8
4.2 Data Dictionary	8

5. COMPONENT DESIGN

14

5.1 ElectionDriver	14
5.2 Election	14
5.3 InstantRunoffElection	15
5.4 OpenPartyListingElection	16
5.5 Candidate	16
5.6 Ballot	16
5.7 Party	17

6. HUMAN INTERFACE DESIGN

17

6.1 Overview of User Interface	17
6.2 Screen Images	18
6.3 Screen Objects and Actions	22

7. REQUIREMENTS MATRIX

23

8. APPENDICES

24

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to provide a detailed description of the Voting System architecture and design. This document is intended for the CSCI 5801 instruction staff at the University of Minnesota. It will also be referenced by our team when implementing the system and possibly making changes.

1.2 Scope

The system will be accessed by programmers, testers and election officials. Programmers will write and test the code. Testers will test code intermittently throughout the design process. Election officials will use the final product to conduct vote tallying. The goal is to produce election results in a clean and timely manner, while maintaining well written documentation and code. More specifically, the key objective is to process 100,000 ballots in under 8 minutes.

1.3 Overview

This document details the architectural design our group decided upon to create a modular program structure. A UML for the system as a whole is provided along with UML activity diagrams for both Instant Runoff voting and Open Party Listing. A sequence diagram detailing Open Party Listing is provided. Data structures and all major data is described in section 4. Section 5 contains pseudocode or summaries of all algorithms and functions. In section 6 an overview of the user interface is provided. Finally, this document and the functionality it describes is cross referenced with our Software Requirement Specifications document to show where use cases are being addressed.

1.4 Reference Material

At various points throughout the document, the Software Requirements Specification document for this system may be referenced. Furthermore, PDF versions of the charts used in this document have been provided for ease of readability and for reference.

1.5 Definitions and Acronyms

All diagrams are produced using UML specifications, where UML stands for Unified Modeling Language. Other than that, there should not be any acronyms that hinder the audience from interpreting this document.

2. SYSTEM OVERVIEW

The Voting System will be able to produce election results for the Instant Runoff voting model and Open Party Listing model. The Instant Runoff model asks voters to rank the candidates in order of preference. If a voter's top preference is eliminated by having the least number of votes, that voter's next best choice receives their vote. Candidates are eliminated until one candidate achieves a majority. In the Open party listing model a voter is able to vote once. That vote counts for one candidate and their party. In Open Party Listing there are a required number of seats to be populated by candidates. Based on the number of seats available and number of ballots cast determines the quota needing to be filled in order for a political party to acquire a seat. If seats remain after counting each party's acquired seats, the remaining seats are awarded to parties based on remaining votes that did not fulfill the quota. Party's seats are given to the candidates with the most votes in each corresponding party.

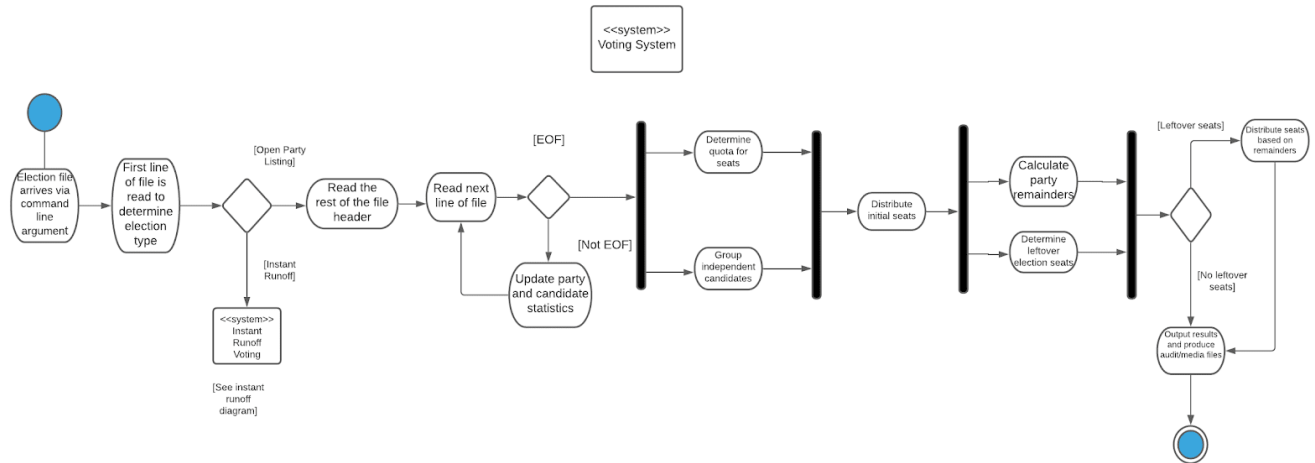
The election in question will have a corresponding ballots CSV file. This file is provided through a command line argument in a terminal when the voting system is run by an election official. At termination a summary of the election will be displayed on screen. Additionally, an audit file with a unique name containing more detailed information about the ballot tallying process will be produced. A media file will be created to provide to media members, containing a summary of the election.

Our team's solution design will require only one program to be run at the terminal to compute both Instant Runoff and Open Party Listing results. The design will feature a modularized model to enhance functionality, readability and efficiency.

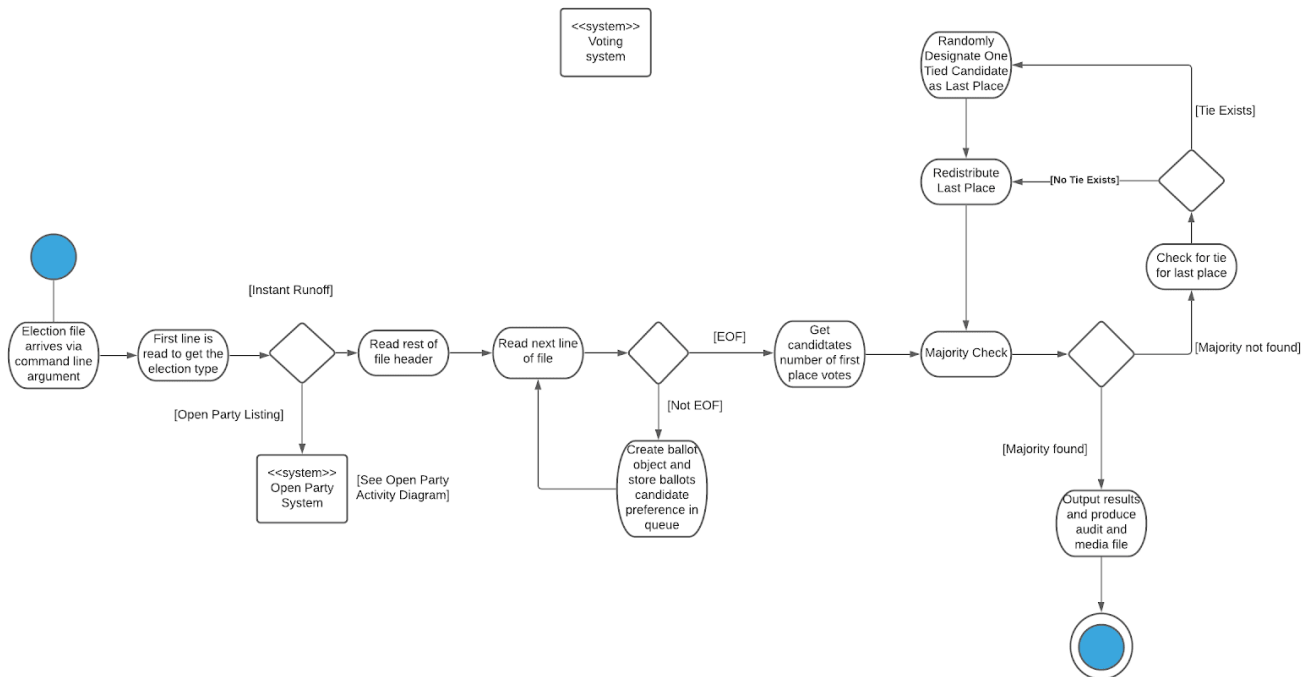
3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The voting system was designed to be able to handle multiple election types within a single program. Hence, the major decompositions occur not necessarily by creating entirely new subsystems, but rather by making use of object oriented programming to decompose elections into basic types such as parties, candidates, and separate elections based on the type denoted in the file. The exact decomposition of our system is described in more detail in Section 3.2 of this document, while the description of all relevant methods is contained within Section 5 of this document. The following UML Activity Diagrams provide a high level overview of the two election types our system will be handling by showing the basic flow of system operation during the course of one of these elections. For a complete description of the algorithms implemented by these election types, please refer to the Software Requirements Specification for the Voting System software.



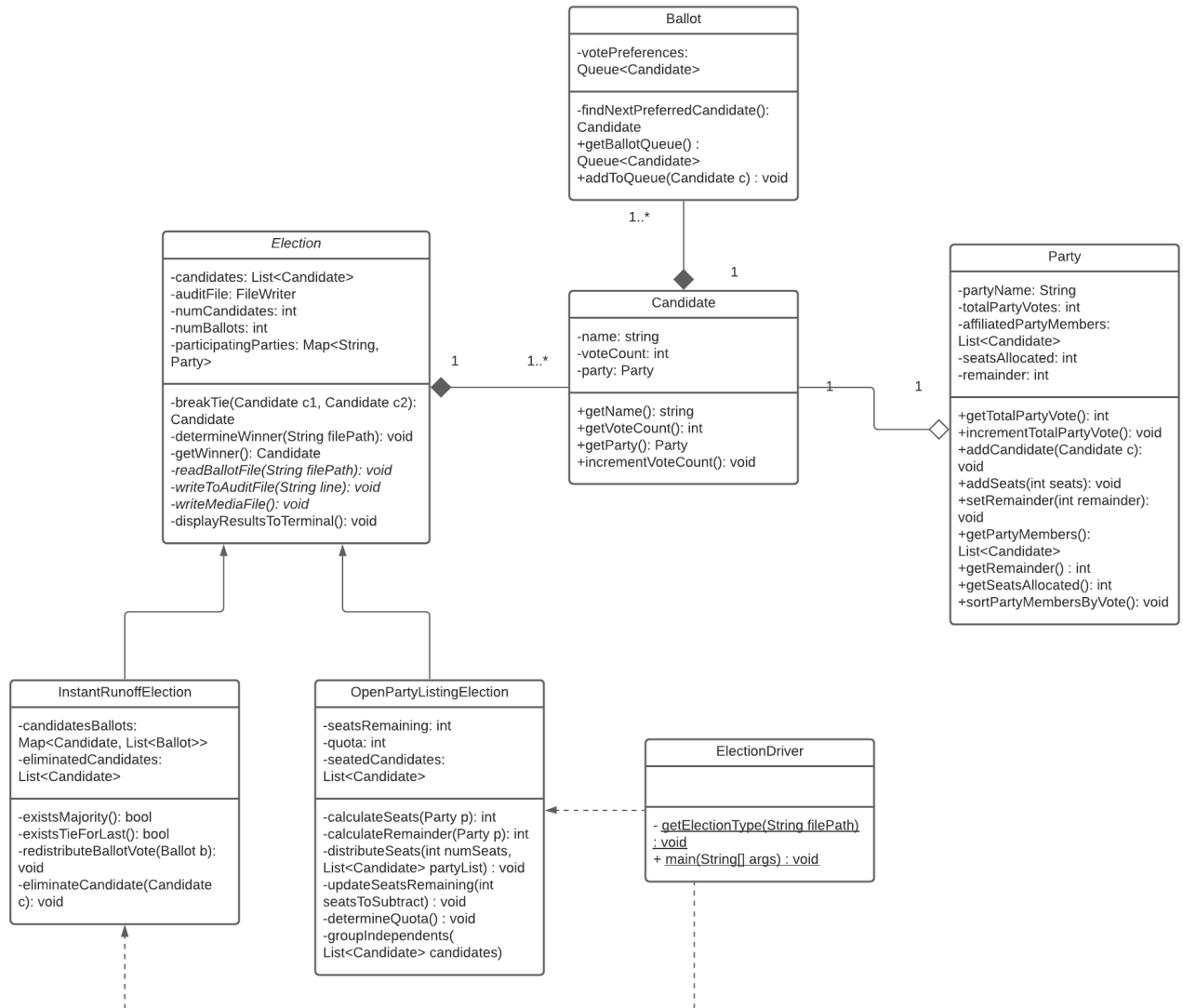
The above figure is the activity diagram displaying the flow of control for the voting system during an Open Party List election. Note that all of these operations are being performed by the Voting System's various objects and methods, hence we have denoted this at the top of the diagram rather than clutter the diagram by linking every single module to the voting system via an arrow. For ease of readability, a PDF version of this diagram is also available within the Software Design Directory that this document is contained within.



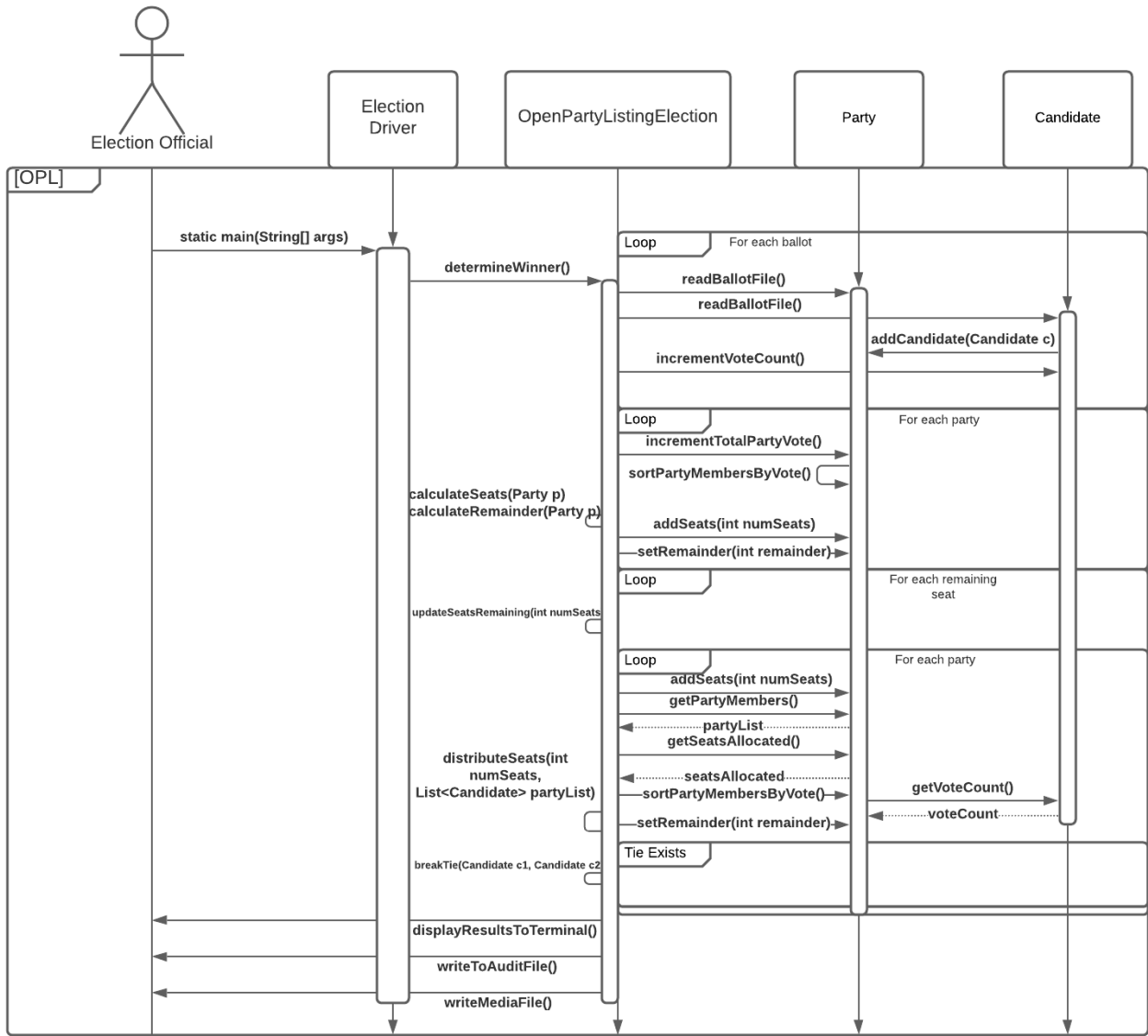
The above figure is the activity diagram displaying the flow of control for the voting system during an Instant runoff election. Note that all of these operations are being performed by the Voting System's various objects and methods, hence we have denoted this at the top of the diagram rather than clutter the diagram by linking every single module to the voting system via an arrow. For ease of readability, a PDF version of this diagram is also available within the Software Design Directory that this document is contained within.

3.2 Decomposition Description

The figure below is the class diagram for our system, as we have chosen to implement an object oriented approach for our system. For ease of readability and clarity, we have also provided a PDF version of this diagram within the same directory as this document.



The following is a sequence diagram for the OpenPartyListing election type as it would be run by our system. For ease of readability and clarity, we have also provided a PDF version of this diagram within the same directory as this document.



3.3 Design Rationale

Our team decided to prioritize an easy to understand and an easily implementable design, while still maintaining an efficient program. We believe that this design will accomplish the objective to complete 100,000 ballots in under 8 minutes. Albeit, we do not believe it to be the most efficient program. There are areas that could see improvements to increase efficiency in both memory and runtime but we believed it would further complicate the design. One such design choice is having two Candidate lists in the Election class and within the Party class. The Candidates list within Election allows for quick access to all the candidates. The Candidate list was kept in the Party class

for quick access to all the affiliated candidates, specifically for Open Party Listing when we wanted to determine seat winners. Implementing both these lists will increase our code's readability. Our initial design included data classes that fed into one Election type class and attempted to reduce extra code for both IR and OPL. This appeared to be more streamlined, although in the end it became more complicated and inefficient than initially thought.

4. DATA DESIGN

4.1 Data Description

Our system chose to encapsulate our data using an object oriented programming approach. Data is stored throughout different classes within their attributes, and methods are used to manipulate these attributes and perform the necessary actions using our data. By using our data in this way, we are able to modularize our system in order to achieve the necessary efficiency while still maintaining the ability to make use of our system for different election types. For example, storing the ballots as objects is a necessary feature for an Instant Runoff election, as they are often manipulated throughout the course of an election, whereas Open Party Listing elections would not require this feature, as ballots only have to be processed one time due to the fact that they do not have to be redistributed.

4.2 Data Dictionary

1. Ballot
 - a. Description: An object to hold the ballot information for an election. Created during the running of an Instant Runoff Election as lines are read from the initial election file.
 - b. Attributes
 - i. votePreferences
 1. Type
 - a. Queue<Candidate>
 - c. Methods
 - i. findNextPreferredCandidate
 1. Parameters
 - a. Void
 2. Return type
 - a. Candidate
 - ii. getBallotQueue
 1. Parameters
 - a. Void
 2. Return type
 - a. Queue<Candidate>
 - iii. addToQueue
 1. Parameters
 - a. Candidate C

- 2. Return type
 - a. Void
- 2. Candidate
 - a. Description: An object representing an individual candidate in an election. Created for each candidate during the parsing of the header of an election file at system startup.
 - b. Attributes
 - i. name
 - 1. Type: string
 - ii. voteCount
 - 1. Type: int
 - iii. party
 - 1. Type: Party
 - iv. preferenceRating
 - 1. Type: int
 - c. Methods
 - i. getName
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. String
 - ii. getVoteCount
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. int
 - iii. getParty
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Party
 - iv. incrementVoteCount
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Void
 - v. setPreferenceRating
 - 1. Parameters
 - a. int i
 - 2. Return type
 - a. void
- 3. Election
 - a. Description: An object representing an individual election. A general class that can be inherited from depending on the type of election that is found when parsing the first line of the file passed into the system at startup.
 - b. Attributes

- i. candidates
 - 1. Type: List<Candidate>
 - ii. auditFile
 - 1. Type: FileWriter
 - iii. numCandidates
 - 1. Type: Int
 - iv. numBallots
 - 1. Type: Int
 - v. participatingParties
 - 1. Type: Map<String, Party>
- c. Methods
 - i. breakTie
 - 1. Parameters
 - a. Candidate c1
 - b. Candidate c2
 - 2. Return type
 - a. Candidate
 - ii. determineWinner
 - 1. Parameters
 - a. String filePath
 - 2. Return type
 - a. Void
 - iii. getWinner
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Candidate
 - iv. readBallotFile
 - 1. Parameters
 - a. String filePath
 - 2. Return type
 - a. Void
 - v. writeToAuditFile
 - 1. Parameters
 - a. String line
 - 2. Return type
 - a. Void
 - vi. writeMediaFile
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Void
 - vii. displayResultsToTerminal
 - 1. Parameters
 - a. Void
 - 2. Return type

- a. Void
- 4. ElectionDriver
 - a. Description: Is the main driver of our system. Has a method to acquire the election type from the file, as well as the main class required to actually run the system/election.
 - b. Attributes
 - i. N/A
 - c. Methods
 - i. getElectionType (static)
 - 1. Parameters
 - a. String filePath
 - 2. Return type
 - a. Void
 - ii. main (static)
 - 1. Parameters
 - a. String[] args
 - 2. Return type
 - a. Void
- 5. InstantRunoffElection
 - a. Description: A class that is a child of the election class. This class represents the Instant Runoff election type in particular, and will be instantiated once the election type is determined. Inherits the methods of the election class described above.
 - b. Attributes
 - i. candidatesBallots
 - 1. Type: Map<Candidate, List<Ballot>>
 - ii. eliminatedCandidates
 - 1. Type: List<Candidate>
 - c. Methods
 - i. existsMajority
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Bool
 - ii. existsTieForLast
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Bool
 - iii. redistribteBallotVote
 - 1. Parameters
 - a. Ballot b
 - 2. Return type
 - a. Void
 - iv. eliminateCandidate
 - 1. Parameters
 - a. Candidate c

2. Return type
 - a. Void
6. OpenPartyListingElection
 - a. Description: A class that is a child of the election class. This class represents the Open Party Listing election type in particular, and will be instantiated once the election type is determined. Inherits the methods of the election class described above
 - b. Attributes
 - i. seatsRemaining
 1. Type: int
 - ii. quota
 1. Type: int
 - iii. seatedCandidates
 1. Type: List<Candidate>
 - c. Methods
 - i. calculateSeats
 1. Parameters
 - a. Party p
 2. Return type
 - a. int
 - ii. calculateRemainder
 1. Parameters
 - a. Party p
 2. Return type
 - a. int
 - iii. distributeSeats
 1. Parameters
 - a. int numSeats
 - b. List<Candidate> partyList
 2. Return type
 - a. Void
 - iv. updateSeatsReamining
 1. Parameters
 - a. int seatsToSubtract
 2. Return type
 - a. Void
 - v. determineQuota
 1. Parameters
 - a. Void
 2. Return type
 - a. Void
 - vi. groupIndependents
 1. Parameters
 - a. List<Candidate> candidates
 2. Return type
 - a. Void

7. Party

- a. Description: A class representing the party that each candidate in an election belongs to. This class will hold statistics relating to each party as a whole, which will be especially useful in the case of the Open Party List election, where party statistics are vital for election proceedings.
- b. Attributes
 - i. partyName
 - 1. Type: String
 - ii. totalPartyVotes
 - 1. Type: int
 - iii. affiliatedPartyMembers
 - 1. Type: List<Candidate>
 - iv. seatsAllocated
 - 1. Type: int
 - v. remainder
 - 1. Type: int
- c. Methods
 - i. getTotalPartyVote
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. int
 - ii. incrementTotalPartyVote
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Void
 - iii. addCandidate
 - 1. Parameters
 - a. Candidate c
 - 2. Return type
 - a. Void
 - iv. addSeats
 - 1. Parameters
 - a. int seats
 - 2. Return type
 - a. Void
 - v. setRemainder
 - 1. Parameters
 - a. int remainder
 - 2. Returns type
 - a. Void
 - vi. getPartyMembers
 - 1. Parameters
 - a. Void
 - 2. Return type

- a. List<Candidate>
- vii. getRemainder
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. int
- viii. getSeatsAllocated
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. int
- ix. sortPartyMembersByVote
 - 1. Parameters
 - a. Void
 - 2. Return type
 - a. Void

5. COMPONENT DESIGN

5.1 ElectionDriver

Description:

The ElectionDriver class serves as the entry point for the program and determines the election type present and instantiates the corresponding election object. The class only contains two member functions-- main() and getElectionType().

Attributes: None

Methods:

- main(String[] args): Acts as the entry point of the program and is passed the ballot file path as a command line argument
- getElectionType(String filePath): Determines whether the election is OPL or IR by opening the file provided and reading the first line. Initializes the corresponding election object

5.2 Election

Description:

The Election class is an abstract class that serves as a template for the OPL and IR Election classes to inherit from. It contains core functionality that both elections need.

Attributes:

- candidates -- List<Candidate>: a list of all candidates in the election
- auditFile -- FileWriter: an object that represents an open audit file for the program to write to
- numCandidates -- int: the total number of candidates
- numBallots -- int: the total number of of ballots
- participatingParties -- Map<String, Party>: a map data structure containing all participating parties as Party objects mapped by their string literal

Methods:

- breakTie(Candidate c1, Candidate c2) -- Candidate: a method that randomly chooses one candidate from the two that are given
- determineWinner(String filePath): serves as the central function of an Election object and calls other methods. It is passed in the ballot file from the ElectionDriver class.
- getWinner(): returns the winner of the election
- readBallotFile(String filePath): reads the ballot file provided and stores the data it receives accordingly
- writeToAuditFile(String line): writes the String passed to the open audit file
- writeMediaFile(): writes the media file using the election data
- displayResultsToTerminal(): writes election data to the terminal

5.3 InstantRunoffElection

Description:

A class dedicated to handling an instant runoff election. It inherits from the abstract class Election.

Attributes:

- candidateBallots -- Map<Candidate, List<Ballot>>: a complex data structure mapping each candidate to a list of ballots assigned to them
- eliminatedCandidates -- List<Candidate>: a list containing all candidates who have been eliminated

Methods:

- existsMajority(): returns true if a candidate has achieved a majority of the votes, false otherwise
- existsTieForLastPlace(): returns true if there is a tie for last place, false otherwise
- redistributeBallotVote(Ballot b): reassigns an eliminated candidate's ballots to the next preferred candidate
- eliminateCandidate(Candidate c): eliminates a candidate from contention

5.4 OpenPartyListingElection:

Description:

A class dedicated to handling an open party listing election. It inherits from the abstract class Election.

Attributes:

- seatsRemaining -- int: the number of seats remaining to be filled
- seatedCandidates -- List<Candidate>: a list of all candidates who have received a seat

Methods:

- calculateSeats(Party p): calculates the total number of seats for a party to receive
- calculateRemainder(Party p): calculates the remainder for the given party
- distributeSeats(int numSeats, List<Candidate> partyList): chooses numSeats number of candidates from the partyList to receive a seat based off of their total votes
- updateSeatsRemaining(int seatsToSubtract): decrements the seatsRemaining attribute by the value given
- determineQuota(): determines the quota based off the total number of ballots and seats to fill
- groupIndependents(List<Candidate> candidates): Groups independent candidates into a single party.

5.5 Candidate:

Description:

A class representing a candidate in the election. Contains information about the candidate.

Attributes:

- name -- String: a candidates' name
- voteCount -- int: the number of votes the candidate has received
- party -- Party: the party the candidate belongs to

Methods:

- getName(): retrieves the candidate's name
- getVoteCount: retrieves the candidate's vote count
- getParty(): retrieves the candidate's party
- incrementVoteCount(): increments the candidate's total vote count by one

5.6 Ballot:

Description:

A class representing a ballot in an election. Contains information about which candidate was voted for.

Attributes:

- votePreferences -- Queue<Candidate>: a priority queue of a ballot's preferred candidates

Methods:

- findNextPreferredCandidate(): moves up in the queue to find the next candidate in line, or, if none exist the ballot is tossed
- getBallotQueue(): returns the votePreferences queue
- addToQueue(Candidate c): adds the provided candidate to the queue

5.7 Party:

Description:

A class representing a political party in an election. Contains information on the party statistics as well as its member candidates

Attributes:

- partyName -- String: the name of the party
- totalPartyVotes -- int: the total number of votes the party has received (in IR only the preliminary votes a party receives are counted towards this number)
- affiliatedPartyMembers -- List<Candidate>: all candidates enrolled in the party
- seatsAllocated -- int: the total number of seats a party has been allocated
- remainder -- int: the total remaining votes after the initial seats have been divvied

Methods:

- getTotalPartyVote(): returns the number of votes the party has received
- incrementTotalPartyVote(): increment the total party votes count by one
- addCandidate(Candidate c): adds the candidate to the affiliated party list
- addSeats(int numSeats): adds number of seats to the total number received thus far
- setRemainder(int remainder): sets the party's remainder value
- getPartyMembers(): returns the affiliated party members list
- getRemainder(): returns the remainder
- getSeatsAllocated(): returns the number of seats a party has been allocated so far
- sortPartyMembersByVote(): sorts the party member list such that members with the most votes are at the front

6. HUMAN INTERFACE DESIGN

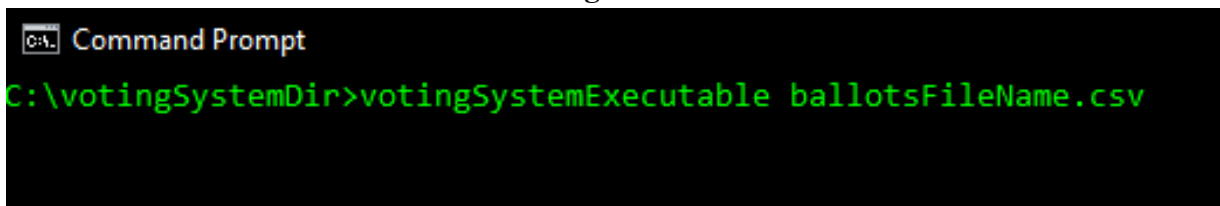
6.1 Overview of User Interface

Election officials and testers will be able to run the Voting System from a terminal on a Linux, Mac or Windows machine with the specified hardware and software described in the System Requirements Specifications. The user should navigate to the correct directory where the program and ballots file are stored. To run the program they will type in the executable name and provide the ballots file name as a command line argument. A confirmation message will be provided if the ballots tallying has successfully begun. If not an error message will appear asking them to re-enter

the file name and to confirm it is within the same directory as the program. While ballots are being tallied, the user may suspend or terminate the system with a Ctrl + C or Ctrl + Z command within the terminal. Signal handling will be handled gracefully, providing a response message with process details prior to the signal. If there is an error within the ballots file itself or any unexpected error is encountered, an error message will be printed to the terminal describing the error and the process will terminate. Upon successful completion of the program a summary of the election results will be printed to the terminal. Additionally, an audit file containing a detailed description of the tallying process will be produced with a unique filename based on the time the program was executed. A media file will be generated to be distributed to media outlets by an election official with information about the election. The user can find the audit file within the same directory as the executable and the ballots file.

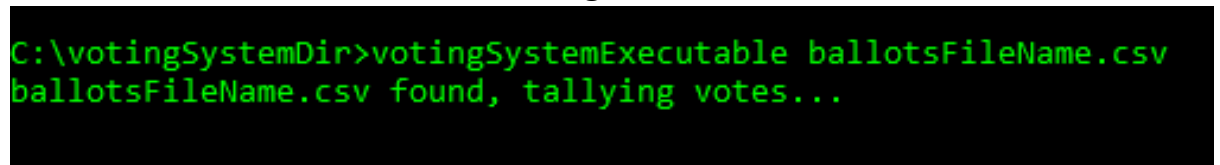
6.2 Screen Images

Figure 1



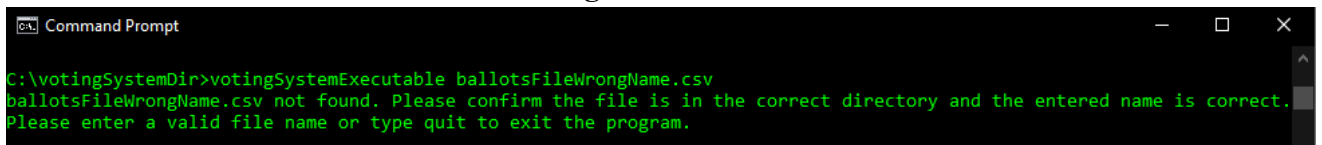
```
C:\votingSystemDir>votingSystemExecutable ballotsFileName.csv
```

Figure 2



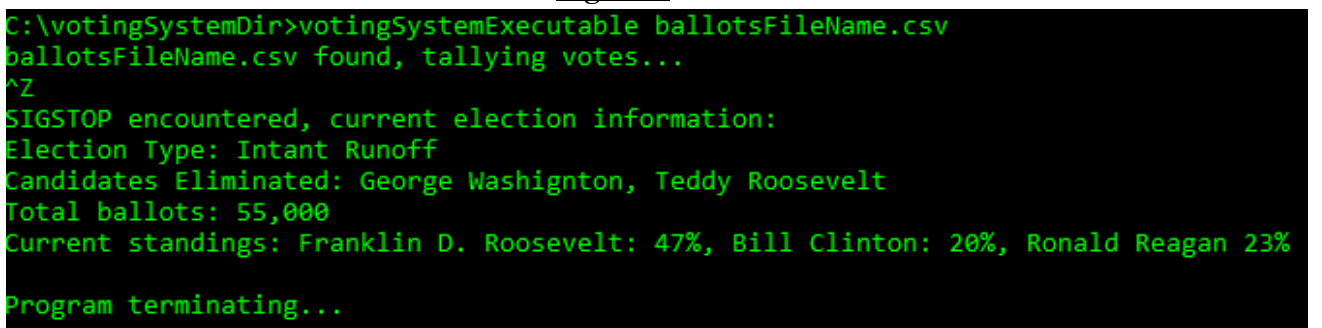
```
C:\votingSystemDir>votingSystemExecutable ballotsFileName.csv
ballotsFileName.csv found, tallying votes...
```

Figure 3



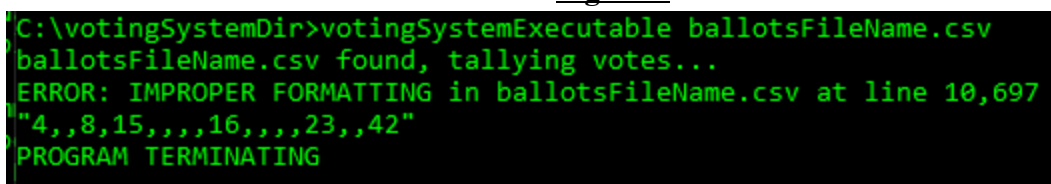
```
C:\votingSystemDir>votingSystemExecutable ballotsFileWrongName.csv
ballotsFileWrongName.csv not found. Please confirm the file is in the correct directory and the entered name is correct.
Please enter a valid file name or type quit to exit the program.
```

Figure 4



```
C:\votingSystemDir>votingSystemExecutable ballotsFileName.csv
ballotsFileName.csv found, tallying votes...
^Z
SIGSTOP encountered, current election information:
Election Type: Instant Runoff
Candidates Eliminated: George Washington, Teddy Roosevelt
Total ballots: 55,000
Current standings: Franklin D. Roosevelt: 47%, Bill Clinton: 20%, Ronald Reagan 23%
Program terminating...
```

Figure 5



```
C:\votingSystemDir>votingSystemExecutable ballotsFileName.csv
ballotsFileName.csv found, tallying votes...
ERROR: IMPROPER FORMATTING in ballotsFileName.csv at line 10,697
"4,,8,15,,,,16,,,,23,,42"
PROGRAM TERMINATING
```

Figure 6

```
C:\votingSystemDir>votingSystemExecutable ballotsFileName.csv
ballotsFileName.csv found, tallying votes...
Program completed successfully.

Election Summary
Election Type: Instant Runoff
Winner: Joe Biden, with 78 % of the vote
Elimination order: (1) Jimmy Carter, (2) Woodrow Wilson, (3) Andrew Jackson
Total Ballots Counted: 99,999

An audit file with the name 2/24/2021-1:20AM has been produced in votingSystemDir
```

Figure 7

Election Type: Instant Runoff
Number of ballots: 99,999
Number of Candidates: 5
Candidates: Joe Biden, Woodrow Wilson, Andrew Jackson, George Bush, Jimmy Carter

No majority

Elim 1:
Joe Biden: 30,000 votes
Woodrow Wilson: 9,999 votes
Andrew Jackson: 30,000 votes
George Bush: 30,000 votes
Jimmy Carter: 0 votes

Jimmy Carter Eliminated, redistributing votes...

Joe Biden: 30,000 + 0 votes
Woodrow Wilson: 9,999 + 0 votes
Andrew Jackson: 30,000 + 0 votes
George Bush: 9,999 + 0 votes

No majority

Elim 2:
Joe Biden: 30,000 votes
Woodrow Wilson: 9,999 votes
Andrew Jackson: 30,000 votes
George Bush: 30,000 votes

Woodrow Wilson Eliminated, redistributing votes...

Joe Biden: 30,000 + 9,998 votes
Andrew Jackson: 30,000
George Bush: 30,000 + 1 vote

No majority

Elim 3:
Joe Biden: 39,998 votes
Andrew Jackson: 30,000 votes
George Bush: 30,001 votes

Andrew Jackson Eliminated, redistributing votes...

Joe Biden: 39,998 + 30,000 votes
George Bush: 30,001 + 0 votes

Majority found, Winner declared:

Joe Biden: 69,998 votes
George Bush: 30,001 votes

Figure 8

Election Type: Open Party Listing
Number of ballots: 40,000
Open Seats: 4
Quota: 10,000 votes
Candidates: John Doe, Jane Smith, John Smith, Christopher Kauffman,
Shana Watters, Conor Brown, Jack Soderwall, Joe Cassidy, Sean Carter
Party Affiliations:

A: John Doe, Jane Smith, Shana Watters
B: Joe Cassidy, Jack Soderwall
C: Conor Brown
D: Sean Carter, Christopher Kauffman

Total votes by party and individual candidates' votes:

A: 21,000
- John Doe(10,000), Jane Smith (1,000), Shana Watters (10,000)
B: 4,000
- Joe Cassidy (1,000), Jack Soderwall (3,000)
C: 10,000
- Conor Brown (10,000)
D: 5,000
- Sean Carter (4,000), Christopher Kauffman (1,000)

Initial Seats Awarded:

A: 2 seats with 1,000 votes remaining
B: 0 seats with 4,000 votes remaining
C: 1 seat with 0 votes remaining
D: 0 seats with 5,000 votes remaining

Awarding 1 seat based on remainder:

A: 0 additional seats with 1,000 votes remaining
B: 0 additional seats with 4,000 votes remaining
C: 0 additional seat with 0 votes remaining
D: 1 additional seats with 0 votes remaining

Winning candidates:

A: John Doe and Shana Watters
B: N/a
C: Conor Brown
D: Sean Carter

Figure 9

```
2/25/2021 Instant Runoff Election Results
-----
Candidates (5): Joe Biden, Woodrow Wilson, Andrew Jackson, George Bush, Jimmy Carter
Number of ballots counted: 99,999
Winner: Joe Biden with majority of 51% of the vote

Elimination order
-----|
(1) Woodrow Wilson, (2) Andrew Jackson, (3) Jimmy Carter

Final votes garnered by each remaining candidate
-----
Joe Biden: 51,000
George Bush: 48,999
```

6.3 Screen Objects and Actions

Figure 1 shows the basic execution of the program in a Windows Terminal. This is how the user starts the program.

Figure 2 shows the initial message produced upon the program successfully finding the file. This confirms to the user that the file was found and tallying has begun. No immediate action is needed.

Figure 3 shows an error message if the file name could not be found. The user should now confirm that the file is in the correct directory and they have entered the correct file name. The user can either enter another file name or type quit to terminate the program.

Figure 4 shows an interaction between the program and a signal stop. If the user decides to stop the program using Ctrl + Z, the program will display the results it found and gracefully terminate. Data will be lost and the program may now be run again from the command line using the initial startup command.

Figure 5 shows an error message printed to the terminal if an error occurs within the ballots file. The user must now fix the file to be able to complete the program and receive election results.

Figure 6 shows successful termination of the program. The user may now take note of the results and inspect the audit file.

Figure 7 is an example of an IR audit file. The user may inspect the audit file for any computation errors or other discrepancies.

Figure 8 shows an example of an OPL audit file. The user may inspect the audit file for any computation errors or other discrepancies.

Figure 9 is an example of a media file that can be distributed to various media outlets by an election official

7. REQUIREMENTS MATRIX

<u>Functional Requirement ID: Name</u>	<u>System Component</u>
VS_01: Obtaining Filename	The main(String[] args) method within the election driver class. The filename will be passed as a command line argument to this method.
VS_02: Display Election Results	The displayResultsToTerminal() method within the Election class.
VS_03: Creating Audit File (IR)	The writeToAuditFile(String line) method within the Election class.
VS_04: Creating Audit File (OPL)	The writeToAuditFile(String line) method within the Election class.
VS_05: File Parsing (IR)	The readBallotFile(String filePath) method of the Election class.
VS_06: File Parsing (OPL)	The readBallotFile(String filePath) method of the Election class.
VS_07: Determining Election Type	The getElectionType() method within the ElectionDriver class.
VS_08: Grouping Independents (OPL)	The groupIndependents(List<Candidate> candidates) method within the OpenPartyListingElection class.
VS_09: Determining Winner (IR)	The determineWinner(String filePath) method within the Election class. Other methods will be called in the process of determining this winner, but this is the primary method driving that process.
VS_10: Determining Winner (OPL)	The determineWinner(String filePath) method within the Election class. Other methods will be called in the process of determining this winner, but this is the primary method driving that process.
VS_11: Breaking Tie Votes	The breakTie(Candidate c1, Candidate c2)

	method within the election class.
VS_12: Creating Media File	The writeMediaFile() method within the election class.

8. APPENDICES

There are no relevant appendices for this document at this time.