

Team 14
Voting System
Software Design Document

Names: Cassandra Chanthamontry (chant077), Jake Nippert (nippe014),
Christine Tsai (tsaix223)

Course: CSCI 5801 Software Engineering I

Date: 10/28/2018

Table of Contents

Table of Contents	2
Revision History	2
1. Introduction	3
1.1 Purpose	3
1.2 Scope	3
1.3 Overview	3
1.4 Reference Material	3
1.5 Definitions and Acronyms	4
2. System Overview	4
3. System Architecture	5
3.1 Architectural Design	5
3.2 Decomposition Description	5
3.3 Design Rationale	7
4. Data Design	8
4.1 Data Description	8
4.2 Data Dictionary	8
5. Component Design	12
5.1 Driver	12
5.2 Interfaces	13
5.3 Auditor	16
5.4 IRV	18
5.5 OPLV	26
6. Human Interface Design	34
6.1 Overview of User Interface	34
6.2 Screen Images	34
6.3 Screen Objects and Actions	36
7. Requirements Matrix	36

Revision History

Name	Date	Reason For Changes	Version
Voting System SDD	October 28, 2018	Version 1.0, first document for system design.	1.0

1. Introduction

1.1 Purpose

This Software Design Document (SDD) describes the implementation of Voting System (VS), specifically the architecture and system design of the software.

The intended audience of this SDD is the election officials as well as testers and software developers who will be implementing and maintaining VS. All parties associated with the elections can also use this document as a reference.

1.2 Scope

This SDD contains a complete description of the implementation of Voting System. Voting System allows the user to produce election results for two main voting structures: Instant Runoff Voting (IRV) and Open Party List Voting (OPLV). Election officials, testers, and software developers will be able to run the system and review the output election results for their respective needs.

The overall goal and objective of the system is to implement all informal user requests so it meets all users' needs and functions efficiently and without error. The system will benefit users by automating the tabulation of election results for both IRV and OPLV, which will reduce the number of human errors and the financial and time inputs required for processing these elections.

1.3 Overview

The first section, Section 2, provides a brief system overview of VS. After this, Section 3 discusses the system architecture including the architectural design, decomposition description, and design rationale. Sections 4, 5, and 6 contain detailed information about the data design, component design, and human interface design, respectively. Section 7 contains a requirements matrix that will cross-reference elements of the SDD to the previously written Software Requirements Specification (SRS) document. Any supporting details will be provided in the Appendices section, Section 8.

1.4 Reference Material

- SRS Document - The SRS document discussing the requirements specifications for VS (SRS_Team14)
- Related diagrams
 - UML class diagram for entire VS (ClassDiagram_Team14)
 - Sequence diagram for running OPLV (SequenceDiagram_Team14)

- UML activity diagram IRV and OPLV (ActivityDiagram_Team14)

1.5 Definitions and Acronyms

- **SDD** - Software Design Document.
- **VS** - Voting System. This is a system where users can provide a csv file with ballot information and choose which voting algorithm to implement to view results of the elections.
- **IRV** - Instant Runoff Voting. Also known as "majority preferential voting." Like two-round voting, this majority system a minor variation of single-member district plurality voting that was developed to ensure that the winning candidate enjoys the support of the majority of voters in the district. It was also thought to be an improvement over the two-round system because it does not require a separate election--it provides an "instant" runoff.
- **PLV** - Party List Voting. Legislators are elected in large, multi-member districts. Each party puts up a list or slate of candidates equal to the number of seats in the district.
- **OPLV** - Open Party List Voting.
- **SRS** - Software Requirements Specification.
- **UML** - Unified Modeling Language.
- **CSV** - comma-separated values file.
- **OS** - Operating System.
- **GUI** - Graphical User Interface.
- **UI** - User Interface.
- **Software** - Programs that determine operations of a system.
- **Hardware** - The physical components of a computer or other electronic system.
- **User** - Refers to testers, software developers, or election officials utilizing Voting System.

2. System Overview

Voting System is designed for election officials to easily analyze results from IRV and OPLV elections. Testers and software developers will also use the software. The program is meant to receive and parse through a provided comma-separated values (CSV) file that contains important details about the election and all the ballots from the election. Once the voting type is determined, the program proceeds to implement the correct algorithm that will calculate the results of the election. As the results are calculated, the system will display relevant information to the user's screen as well as outputting an audit file that will contain more details about how the election proceeded throughout the algorithm.

3. System Architecture

3.1 Architectural Design

3.1.1 Class Diagram

Refer to document ClassDiagram_Team14.pdf.

The class diagram demonstrates the relationship between the Interfaces and Classes. The Driver class is an association of the Voting System interface and the Auditor is a composition of the Voting System interface. For the association between the Driver and the Voting System, only one driver can exist per VotingSystem. For the composition relationship between the Voting System and the Auditor, there is one Auditor per Voting System. The IRV and OPLV classes are realizations of the Voting System interface.

With the OPLV class, the OPLVBallot, OPLVCandidate, and Party classes are compositions of it. In this composition relationship the OPLVBallot, OPLVCandidate and Party can only belong to the one OPLV class. However, an OPLV class can have one to many OPLVBallots, OPLVCandidates and Parties. OPLV also has an aggregated relation with OPLVCandidates such that these candidates make up seats in the voting system as winners of the election. As an aggregation, the OPLVCandidate must not belong to a seat and can only make up one seat if selected as a winner. The OPLV will have one to many seats in the elections. The OPLVBallot is a realization of the Ballot Interface and the OPLVCandidate is a realization of the Candidate Interface.

With the IRV class, the IRVCandidate and IRVBallot are compositions of it. For the IRV and IRVCandidate relationship, there can be one to many IRVCandidate per IRV. The IRV and IRVBallot follow a similar relationship, where there can be one to many IRVBallot per IRV. IRVCandidate has an aggregated relationship with IRVBallot. For one IRVCandidate there can be zero to many IRVBallots. The IRVBallot is a realization of the Ballot Interface and the IRVCandidate is a realization of the Candidate Interface.

3.2 Decomposition Description

3.2.1 Activity Diagram of IRV and OPLV

Refer to document ActivityDiagram_Team14.pdf.

Activity Diagram Overview:

The first step is to check that the user provided a CSV file. If the file does not exist in the program directory, the system will output an error message to the terminal and continue to prompt the user to provide a filename. If the file does exist, it is opened and the voting type is checked. If the voting type is OPLV, the program runs through the OPLV algorithm. Otherwise, it proceeds with the IRV algorithm.

For the OPLV voting, the system concurrently retrieves the number of candidates, seats, and ballots. After retrieving the number of candidates, it retrieves each candidate and their respective parties. The quota is then calculated. If a ballot is unassigned, it is assigned to a candidate and party and then the assignment is recorded to the auditor. Once there are no unassigned ballots left, each party's candidates are ordered by popularity and the number of seats per party is calculated. The remaining seats are allocated to parties with the largest remainders. Seats are filled for each party based on the popularity of candidates. The seat assignments are recorded to the auditor and then the winners and election information are output to the terminal. The auditor creates an output file and then the program is terminated.

For IRV voting, the number of candidates and the number of ballots are retrieved. The quota for the majority is determined. Then if all voter pool ballots have been processed, the voter pool is updated to the candidate with the least number of votes. The elimination is recorded to the auditor. If there is an unprocessed ballot in the pool, then the system assigns the ballot to a candidate based on n-ary vote from the ballot. The assignment is recorded to the auditor. The system then checks if a candidate has a majority based on the quota. If there is a majority winner, the winner is recorded to the auditor. Finally, the winner and election information is output to the terminal. The auditor creates an output file and then the program is terminated. If there is no majority winner, the system repeats the steps starting from updating the voter pool with the candidate with the least number of votes.

All the number one preferences are added up. The vote counts are checked to see if there was a majority vote winner. If so, the winner and election information is output to the terminal. A file containing an audit of the election is created and placed in the same directory containing the program. The program then closes, ending the process. If there is no majority vote winner, the system checks if all votes have been handed out. If so, then the popular vote is set as the winner and the system proceeds through the same output

3.2.2 Sequence Diagram for OPLV

Refer to document SequenceDiagram_Team14.pdf

Sequence Diagram Overview:

The user runs the program which starts main in the Driver. One an input file name is provided, the driver then constructs the OPLV voting system by providing the ballot file. The voting system constructor is then called to implement the voting system interface. The auditor is then constructed which maintains information and produces the audit file. From here the OPLV must start processing the information before the election can be run. For every candidate in the provided file a candidate constructor must be called with the name and party. If the party does not exist then the party constructor is called from the OPLV with the name, the subsequent object is tied to the candidate in its constructor, and the candidate is then added to the party. The party is then returned to the OPLV voting system along with the candidate. If the party does exist then the candidate constructor is called with the party as normal and the candidate object is returned. Now all ballots must be processed for their information; this entails collecting the index of the ballot vote as well as a system generated ballot id. The ballot is then returned to the OPLV. The OPLV can then be returned to the Driver as fully constructed from which the User can run the election. When an election is run the quota is calculated as a function of the number of ballots and number of seats. We then loop through all ballots getting their votes, returning the index, finding the candidate with that index and casting a vote for them to the OPLVCandidate. Since a vote for the candidate is a vote for the party we must add a vote to the Party. Finally we send this information from the OPLV to the auditor to be maintained. When this loop is exited, we must rank all of the parties. We enter another loop which for each party ranks all of the candidates, tracking instances of randomized ties which are then sent to the auditor per party ranking. After this loop we calculate party seats to determine how many seats will be assigned to each party; this is done through a largest remainder formula. We intentionally do not include setting the remainders and retabulating the seats. Once this is completed the OPLV must assign seats by looping through all Parties and returning the winners as ranked and allocated by the number of seats available. These seats are sent to the auditor to be maintained. The winners are then sent back to the Driver which will print them to the terminal. We intentionally do not include the audit file's creation of the audit file which will be a conglomeration of all election information.

3.3 Design Rationale

3.3.1 Efficiency

One of the requirements was to process 100,000 votes in 8 seconds. To accommodate this functionality we made an enumerated attribute that store the candidates names indexed by a number. Therefore we can quickly find candidates.

3.3.2 Unincorporated Architecture

We chose to not store the ballot votes as a string as originally planned due to the fact that comparing strings is significantly less efficient than comparing integer values. By storing the vote

as the index value of the candidate in the candidates array attribute we can easily retrieve the candidate that each ballot intended. We also chose to not utilize a Seat class and instead make this a relationship between candidates in OPLV and the VotingSystem itself. This reduced the amount of dependence on inefficient class constructions.

3.3.3 Justification of Classes

We chose to utilize interfaces for candidates and ballots as many attributes are shared among the OPLV and IRV candidate and ballot classes. We tried to utilize OOP concepts such as encapsulation so that each miniworld object has its own class with attributes and methods private to it that are internal. The public methods can be accessed for most classes mainly by the VotingSystem class which is composed from most other classes in our system. We chose to utilize compositions here as these ballots, candidates, parties and even the auditor exist only as a part of the voting system utilized in our system. We could eliminate many redundancies by using interfaces and reduce the amount of code and development time. When classes communicate we try to eliminate the passing of objects and tend to rather use integers and strings that are understood by the receiving party, this was certainly a security consideration.

3.3.4 Functionality

The methods developed for classes, especially private methods. Allow for code reuse, generalizability, and future code improvements or abstractions. Furthermore, by only making well thought out methods and data public, much of the data and functionalities are hidden from the User that should not be utilized by the user.

4. Data Design

4.1 Data Description

The information we parse from the Ballot CSV File that was given from the user is transformed into objects. Namely, Voting System, Candidates, Ballots, and Party. The Auditor object is developed as we go. We separated the auditing to have an external class to track the major steps in the algorithm we will run to determine the election results. Another key data structure is that the Voting System are compositions of many major elements that we separated into classes. Namely, the Candidates, Ballots, Parties, and Auditor. The major data structure is the Voting System. The Voting System and Auditor have a close relationship.

4.2 Data Dictionary

<<VotingSystem>>

Data Item	Data Type	Description
auditor	Auditor	An auditor keeps track of all

		determinations in the algorithmic determination of election results.
numBallots	int	Number of ballots in the election. Derived from ballot file.
numCandidates	int	Number of candidates participating in the election. Derived from ballot file.
quota	int	The minimum number of votes necessary to secure a win (IRV majority) or seat (OPLV).

IRV

Data Item	Data Type	Description
ballots	IRVBallots[]	A list of all ballots in the election. Derived from ballot file.
candidates	IRVCandidate[]	An ordered list of all candidates participating in the election. Derived from ballot file.
voterPool	IRVBallot[]	A list of all ballots in the pool of voters currently being processed.

OPLV

Data Item	Data Type	Description
ballots	OPLVBallots[]	A list of all ballots in the election. Derived from ballot file.
candidates	IRVCandidate[]	An ordered list of all candidates participating in the election. Derived from ballot file.
seats	OPLVCandidate[]	A list of winners of an

		election.
numParties	int	Number of parties involved in the election. Derived from ballot file.
numSeats	int	Number of seats available in the election. Derived from ballot file.
parties	Party[]	A list of all parties in the election. Derived from ballot file.

Auditor

Data Item	Data Type	Description
auditProcess	String	A string of all determinations in the algorithm in the process of determining the winner(s).
auditResult	String	A string of the result of the election.

<<Candidate>>

Data Item	Data Type	Description
name	String	Name of the candidate.
numVotes	int	The number of votes the candidate has received in the election.

IRVCandidate

Data Item	Data Type	Description
elimBallots	IRVBallot[]	The ballots associated with the candidates who are not exhausted and will become the voting pool if the candidate is eliminated.

OPLCandidate

Data Item	Data Type	Description
party	Party	The party associated with the candidate.

<<Ballot>>

Data Item	Data Type	Description
id	int	A unique identifier for the ballot. System generated.

IRVBallot

Data Item	Data Type	Description
votes	int[]	A list of votes in order by index in the candidates array. Derived from the ballot file.
curVoteIndex	int	Keeps track of the index in votes array the ballot will be cast for next.
numVotes	int	Keeps track of the maximum number of votes associated for the specific ballot.

OPLVBallot

Data Item	Data Type	Description
vote	int	An index in the candidates array that the ballot is voting for.

Party

Data Item	Data Type	Description
candidates	OPLVCandidate[]	A list of all candidates that belong to the party. Derived from the file.
name	String	The name of the party.

		Derived from the file.
numCandidates	int	The number of candidates that belong to the party.
numSeats	int	The number of seats the party has been allocated.
numVotes	int	The number of derived votes the party has received from a vote cast to a respective candidate.

Please refer to section five below for entity methods, parameters and descriptions.

5. Component Design

5.1 Driver

5.1.1 Driver Class

5.1.1.1 main

Purpose	Collects user input and runs the election
Prototype	public static void main()
Inputs	Filename <optional>
Outputs	None
Called By	None
Calls	IRV() or OPLV(), VotingSystem.runElection()
Flow of Events	<pre> File ballotFile = open(user_input()) String votingSystem = ballotFile.getLine() votingSystem = NULL if (votingSystem == 'IRV'): votingSystem = IRV(ballotFile) else: VotingSystem = OPLV(ballotFile) String winners = votingSystem.runElection() print(winners) </pre>

5.2 Interfaces

5.2.1 Voting System Interface

5.2.1.1 VotingSystem

Purpose	Abstract Constructor for VotingSystem
Prototype	protected VotingSystem(File ballotFile)
Inputs	Name of the ballot file
Outputs	None
Called By	IRV(), OPLV()
Calls	Auditor()
Flow of Events	set numBallots, numCandidates and quota Auditor auditor = Auditor()

5.2.1.2 runElection

Purpose	Run the election, print the results and create audit file
Prototype	public String runElection()
Inputs	None
Outputs	String of election winner(s)
Called By	Driver.main()
Calls	N/A <<interface>>: Implementation Dependent
Flow of Events	N/A <<interface>>: Implementation Dependent

5.2.1.3 auditResults

Purpose	Send results of election to auditor
Prototype	private void auditResults(String results)

Inputs	String of election results
Outputs	None
Called By	runElection()
Calls	Auditor.result()
Flow of Events	auditor.result(results)

5.2.2 Candidate Interface

5.2.2.1 Candidate

Purpose	Constructor for Candidate interface
Prototype	protected Candidate(String inputName)
Inputs	Name of candidate
Outputs	None
Called By	IRVCandidate(), OPLVCandidate()
Calls	None
Flow of Events	String name = name int numVotes = 0

5.2.2.2 getName

Purpose	To securely get the name of the Candidate
Prototype	public string getName()
Inputs	None
Outputs	The string name of the candidate
Called By	IRV.runElection(), IRV.findCandidate(), IRV.processVoterPool(), IRV.popularVote(), OPLV.runElection(), OPLV.findCandidate(),
Calls	None
Flow of Events	return candidate.Name

5.2.2.3 getNumVotes

Purpose	To securely get the number of votes
Prototype	public int getNumVotes()
Inputs	None
Outputs	The int number of votes for the candidate
Called By	IRV.findMinimumCandidate(), IRV.popularVote()
Calls	None
Flow of Events	return candidate.NumVotes

5.2.3 Ballot Interface**5.2.3.1 Ballot**

Purpose	Constructor for the Ballot interface
Prototype	protected Ballot(int inputID)
Inputs	Id of the ballot
Outputs	None
Called By	IRVBallot(), OPLVBallot()
Calls	None
Flow of Events	int id = inputID

5.2.3.2 getID

Purpose	To get the ID of the certain ballot
Prototype	public int getID()
Inputs	None
Outputs	The int ID of the ballot
Called By	IRV.processVoterPool(), OPLV.runElection()

Calls	None
Flow of Events	return ballot.ID

5.3 Auditor

5.3.1 Auditor Class

5.3.1.1 Auditor

Purpose	Constructor for the Auditor class
Prototype	Auditor()
Inputs	None
Outputs	None
Called By	VotingSystem.VotingSystem()
Calls	None
Flow of Events	String auditProcess = '' String auditResult = ''

5.3.1.2 ballot

Purpose	Records the assignment of a ballot to a given candidate (and party for OPLV)
Prototype	public ballot(int ballotID, String candidate, String party = '')
Inputs	Ballot ID, candidate name, and party name
Outputs	None
Called By	VotingSystem.runElection()
Calls	None
Flow of Events	auditProcess += String.format("Ballot %d cast a vote for %s", ballotID, candidate) if (party != ''): auditProcess += String.format("in party %s", party)

	auditProcess += "\n"
--	----------------------

5.3.1.3 eliminateCandidateIRV

Purpose	Record the elimination of a candidate in the instant runoff voting system
Prototype	public void eliminateCandidateIRV(String candidate, int numVotes, int[] ballotIDs, bool wasRandom)
Inputs	Name of candidate, their number of votes, the ballot IDs associated with the candidate, and whether there was a random decision to eliminate this candidate due to a tie
Outputs	None
Called By	VotingSystem.runElection()
Calls	None
Flow of Events	<pre> auditProcess += String.format("Candidate %s was eliminated with only %d votes from the following ballots:\n", candidate, numVotes) for bal in ballotIDs: auditProcess += String.format("\t%d\n", bal) if (wasRandom): auditProcess += ("The elimination of this candidate was randomly decided due to a tie in voting with one or more other candidates\n") </pre>

5.3.1.4 rankOPLV

Purpose	Audits the ranking of individual parties
Prototype	public void rankOPLV(String rankMsg)
Inputs	String of the ranking message
Outputs	None
Called By	VotingSystem.runElection()
Calls	None
Flow of Events	auditProcess += rankMsg

5.3.1.5 result

Purpose	Audits the results of the election
Prototype	public void result(String results)
Inputs	String of the results of the election
Outputs	None
Called By	VotingSystem.runElection()
Calls	None
Flow of Events	auditResults = results

5.3.1.6 createAuditFile

Purpose	Outputs the audit information to a file
Prototype	public void createAuditFile(String name)
Inputs	Name of the audit file to store on the machine
Outputs	None
Called By	VotingSystem.runElection()
Calls	None
Flow of Events	File file = new File(name) FileWriter writer = new FileWriter(file) writer.write(auditProcess + "\nElection Results:\n" + auditResults) writer.close()

5.4 IRV**5.4.1 IRV Class : VotingSystem****5.4.1.1 runElection**

Purpose	This is the main algorithm for how an instant runoff voting system runs an election
Prototype	public String runElection()

Inputs	None
Outputs	String of election winner name
Called By	Driver.main()
Calls	processVoterPool(), findMinimumCandidate(), Auditor.eliminateCandidateIRV(), Candidate.getName(), IRVCandidate.eliminate(), IRVBallot.isExhausted()
Flow of Events	<pre> while (True): allBallotsExhausted = True for bal in ballots: if (bal.isExhausted()) allBallotsExhausted = False break if (allBallotsExhausted): return popularVote(): String winner = processVoterPool() if (winner != ""): return winner else: <IRVCandidate, bool> minRes = findMinimumCandidate() auditor.eliminateCandidateIRV(minRes.first().getName(), minRes.second()) voterPool = minRes.first().eliminate() </pre>

5.4.1.2 IRV

Purpose	Constructor for IRV by parsing ballot file
Prototype	IRV(File ballotFile)
Inputs	Ballot File to parse
Outputs	None
Called By	Driver.main()
Calls	Ballot(), IRVBallot(), IRVCandidate()
Flow of Events	Super(ballotFile) #VotingSystem constructor get and construct ballots and candidates from file voterPool = ballots

5.4.1.3 calculateQuota

Purpose	Set the quota necessary to win election
Prototype	private int calculateQuota(int numBallots)
Inputs	The int total number of ballots for the IRV election
Outputs	The quota needed to win the election
Called By	Driver.main()
Calls	None
Flow of Events	<pre> if ((int numBallots*0.5)%2) == 0: quota = (numBallots*0.5)+1 else: quota = ceil(int numBallots * 0.5) return quota </pre>

5.4.1.4 findCandidate

Purpose	Finds a candidate given the candidate name
Prototype	private IRVCandidate findCandidate(int index)
Inputs	Name of candidate
Outputs	IRVCandidate at the index passed in
Called By	processVoterPool()
Calls	None
Flow of Events	return candidates[index]

5.4.1.5 findMinimumCandidate()

Purpose	Finds the candidate with the minimum number of votes
Prototype	private <IRVCandidate, bool> findMinimumCandidate()
Inputs	None

Outputs	Pair of candidate and boolean indicating if random decision
Called By	IRV.runElection()
Calls	None
Flow of Events	<pre> IRVCandidate[] minCandidates = [] minimum = candidates[0].getNumVotes() for can in candidates: if (can.getNumVotes == minimum): minCandidates.append(can) elif (can.getNumVotes() < minimum): minCandidates = [can] minimum = can.getNumVotes() if (minCandidates.length() == 1): return <minCandidates[0], False> else: return <Random(minCandidates), True> </pre>

5.4.1.6 setVoterPool

Purpose	Sets the voter pool to the input set of ballots
Prototype	private void setVoterPool(IRVBallot[] ballots)
Inputs	List of ballots
Outputs	None
Called By	IRV.runElection()
Calls	None
Flow of Events	voterPool = ballots

5.4.1.7 processVoterPool

Purpose	Process the voter pool ballots in election run
Prototype	private String processVoterPool()
Inputs	None
Outputs	String of winning candidate or empty string if no majority reached

Called By	IRV.runElection()
Calls	findCandidate(), IRVBallot.getNextVote(), IRVCandidate.addBallot(), Auditor.ballot(), IRVBallot.getID(), IRVCandidate.getName(), IRVCandidate.isMajority()
Flow of Events	<pre> for bal in voterPool: IRVCandidate can = findCandidate(bal.getNextVote()).addBallot(bal) auditor.ballot(bal.getID(), can.getName()) if (can.isMajority()): return can.getName() </pre>

5.4.1.7 popularVote

Purpose	Determine the popular vote if not clear majority
Prototype	private String popularVote()
Inputs	None
Outputs	String of winning candidate
Called By	IRV.runElection()
Calls	
Flow of Events	<pre> maximum = 0 String[] winners = [] for can in candidates: if can.getNumVotes() > maximum: maximum = can.getNumVotes() winner = [can.getName()] elif can.getNumVotes() == maximum: winners.append(can.getName()) if (winners.length() == 1): return winners[0] else: return Random(winners) + "chosen at random for tied popular vote\n" </pre>

5.4.2 IRVCandidate Class : Candidate

5.4.2.1 IRVCandidate

Purpose	Constructor for IRVCandidate class
Prototype	IRVCandidate(String name)
Inputs	Name of candidate
Outputs	None
Called By	IRV.IRV()
Calls	None
Flow of Events	Super(name) #Candidate constructor IRVBallot[] ballots = [] bool eliminated = False

5.4.2.2 addBallot

Purpose	Adds a ballot to a candidate then increases number of votes for candidate
Prototype	public void addBallot(IRVBallot ballot)
Inputs	Ballot assigned to candidate
Outputs	None
Called By	IRV.processVoterPool()
Calls	None
Flow of Events	if (!ballot.isExhausted()): elimBallots.append(ballot) numVotes += 1

5.4.3.3 isMajority

Purpose	Checks if a candidate has a majority of votes based on quota
Prototype	public bool isMajority(int quota)
Inputs	Quota of sufficient votes for a majority
Outputs	Boolean of whether the candidate has a majority of votes

Called By	IRV.processVoterPool()
Calls	None
Flow of Events	return numVotes >= quota

5.4.3.5 eliminate

Purpose	Indicate that a candidate has been eliminated and return their voters
Prototype	public IRVBallot[] eliminate()
Inputs	None
Outputs	A list of ballots who votes for this candidate
Called By	IRV.runElection()
Calls	None
Flow of Events	eliminated = False return elimBallots

5.4.4 IRVBallot Class : Ballot

5.4.4.1 IRVBallot

Purpose	Constructor for IRVBallot
Prototype	IRVBallot(int[] inputVotes, int id)
Inputs	String of all ballot votes, id of ballot
Outputs	None
Called By	IRV.IRV()
Calls	None
Flow of Events	Super(id) # Constructor of Ballot int[] votes = votes int curVoteIndex = 0

5.4.4.2 getNextVote

Purpose	To securely get the next vote from the ballot
Prototype	public string getNextVote()
Inputs	None
Outputs	The string of the next vote in the votes string array
Called By	IRV.processVoterPool()
Calls	None
Flow of Events	curVoteIndex += 1 return votes[curVoteIndex - 1]

5.4.4.2 isExhausted

Purpose	Determines if a ballot has more votes to offer
Prototype	public bool isExhausted()
Inputs	None
Outputs	Boolean for if a ballot has more votes to offer
Called By	IRV.runElection(), IRVCandidate.addBallot()
Calls	None
Flow of Events	return votes.length() <= curVoteIndex

5.5 OPLV

5.5.1 OPLV Class : VotingSystem

5.5.1.1 runElection

Purpose	This is the main algorithm for how an open party list voting system runs an election
Prototype	public String runElection()
Inputs	None

Outputs	String of election winners
Called By	Driver.main()
Calls	findCandidate(), OPLVBallot.getVote(), OPLVCandidate.castVote(), Auditor.ballot(), Ballot.getID(), OPLVCandidate.getName(), OPLVCandidate.getParty(), Party.getName()
Flow of Events	<pre> for bal in ballots(): OPLVCandidate can = findCandidate(bal.getVote()).castVote() auditor.ballot(bal.getID(), can.getName(), can.getParty().getName()) rankPartyCandidates() calculatePartySeats() assignSeats() </pre>

5.5.1.2 OPLV

Purpose	Constructor for OPLV by parsing ballot file
Prototype	OPLV(File ballotFile)
Inputs	Ballot File to parse
Outputs	None
Called By	Driver.main()
Calls	Ballot(), OPLVBallot(), OPLVCandidate()
Flow of Events	<pre> Super(ballotFile) #VotingSystem constructor get and construct ballots, candidates, parties from file set numParties, numSeats </pre>

5.5.1.3 calculateQuota

Purpose	To set the quota necessary for winning a seat
Prototype	private calculateQuota(numBallots, numSeats)
Inputs	The total number of ballots in the election and the number of seats up for election
Outputs	The quota for winning a seat
Called By	Driver.main()

Calls	None
Flow of Events	<pre> if ((int numBallots*0.5)%2) == 0: quota = (numBallots*0.5)+1 else: quota = ceil(int numBallots * 0.5) return quota </pre>

5.5.1.4 rankPartyCandidates

Purpose	Rank the candidates for all parties
Prototype	private void rankParty()
Inputs	None
Outputs	None
Called By	OPLV.runElection()
Calls	Party.rankCandidates(), Auditor.rankOPLV()
Flow of Events	<pre> for party in parties: String rankMsg = party.rankCandidates() auditor.rankOPLV(rankMsg) </pre>

5.5.1.5 calculatePartySeats

Purpose	Calculate the number of seats each party is assigned
Prototype	private void calculatePartySeats()
Inputs	None
Outputs	None
Called By	OPLV.runElection()
Calls	Party.setNumSeats(), Party.getNumSeats()
Flow of Events	<pre> int[] remainder = [] for party in parties party.setNumSeats(RoundDown(party.getNumVotes() / quota)) remainders.append(party.getNumVotes() % quota) indx = LargestIndex(remainders) </pre>

	<code>parties[indx].setNumSeats(parties[indx].getNumSeats() + 1)</code>
--	---

5.5.1.6 assignSeats

Purpose	Assign the final array of election winners in seats
Prototype	<code>private void assignSeats()</code>
Inputs	None
Outputs	None
Called By	<code>OPLV.runElection()</code>
Calls	<code>Party.getWinningCandidates()</code>
Flow of Events	for party in parties seats.append(party.getWinningCandidates())

5.5.1.7 findCandidate

Purpose	Finds a candidate given the candidate name
Prototype	<code>private OPLVCandidate findCandidate(int index)</code>
Inputs	Name of candidate
Outputs	OPLVCandidate at the index passed in
Called By	<code>OPLV()</code>
Calls	None
Flow of Events	<code>candidates[index]</code>

5.5.2 OPLVCandidate Class : Candidate

5.5.2.1 castVote

Purpose	Casts a vote to a given candidate from a ballot
Prototype	<code>public void castVote()</code>
Inputs	None

Outputs	None
Called By	OPLV.runElection()
Calls	None
Flow of Events	numVotes += 1

5.5.2.2 OPLVCandidate

Purpose	Constructor for OPLVCandidate
Prototype	OPLV(String name, Party inputParty)
Inputs	Name of candidate and the Party the candidate is a part of
Outputs	None
Called By	OPLV.OPLV()
Calls	Candidate.Candidate()
Flow of Events	Super(name) # Constructor of candidate party = inputParty

5.5.2.3 getParty

Purpose	To get the party that the candidate is associated with
Prototype	public string getParty()
Inputs	None
Outputs	The string party name of the candidate
Called By	OPLV.runElection()
Calls	None
Flow of Events	return party

5.5.3 OPLVBallot Class : Ballot

5.5.3.1 OPLVBallot

Purpose	Constructor for OPLVBallot
Prototype	OPLVBallot(String vote, int id)
Inputs	String of ballot vote, id of ballot
Outputs	None
Called By	IRV.IRV()
Calls	None
Flow of Events	Super(id) # Constructor of Ballot String[] votes = votes int curVoteIndex = 0

5.5.3.2 getVote

Purpose	To securely get the votes
Prototype	public string getVote()
Inputs	None
Outputs	Output what candidate is voted for
Called By	OPLV.runElection()
Calls	None
Flow of Events	return vote

5.5.4 Party Class

5.5.4.1 Party

Purpose	Constructor for party
Prototype	Party(String inputName)
Inputs	Name of the party
Outputs	None
Called By	OPLV()

Calls	None
Flow of Events	OPLVCandidate[] candidates = [] String name = inputName Set numCandidates, numSeats and numVotes = 0

5.5.4.2 addCandidate

Purpose	Adds a candidate to the party when being initialized
Prototype	public void addCandidate(OPLVCandidate candidate)
Inputs	Candidate being added to the party
Outputs	None
Called By	OPLV()
Calls	None
Flow of Events	candidates.append(candidate) numCandidate += 1

5.5.4.3 addVote

Purpose	Adds a vote to the party when a ballot for a candidate with this party is voted for
Prototype	public void addVote
Inputs	None
Outputs	None
Called By	OPLV.runElection()
Calls	None
Flow of Events	numVotes += 1

5.5.4.4 getName

Purpose	To securely get the name of the Party
---------	---------------------------------------

Prototype	public string getName()
Inputs	None
Outputs	The string name of the party
Called By	OPLV.runElection()
Calls	None
Flow of Events	return name

5.5.4.5 getNumVotes

Purpose	To securely get the number of votes
Prototype	public int getNumVotes
Inputs	None
Outputs	The int number of votes for the party
Called By	OPLV.calculatePartySeats()
Calls	None
Flow of Events	return numVotes

5.5.4.6 setNumSeats

Purpose	To set the number of seats available for the party
Prototype	public setNumSeats(int inputNumSeats)
Inputs	The number of seats available for that party
Outputs	None
Called By	OPLV.calcualtePartySeats()
Calls	None
Flow of Events	numSeats = inputNumSeats

5.5.4.7 getNumSeats

Purpose	To securely get the number of seats available for the party
Prototype	public int getNumSeats()
Inputs	None
Outputs	The number of seats available for that party
Called By	OPLV.calculatePartySeats()
Calls	None
Flow of Events	return numSeats

5.5.4.8 rankCandidates

Purpose	Ranks the candidates in the party and returns a string of ranking with an indication of randomization was utilized
Prototype	public String rankCandidates()
Inputs	None
Outputs	String of all candidates in order with randomization message if applicable
Called By	OPLV.rankPartyCandidates()
Calls	None
Flow of Events	<pre>Sort(candidates, getNumVotes()) # with randomized ties int prev = -1 String msg = String.format("Ordered candidates for party %s:\n", name) for can in candidates[0..]: if (can.getNumVotes() == prev): random = True msg += can.getName() + "\n" if random: msg += "Randomization of candidates with equal votes occurred" return msg</pre>

5.5.4.9 getWinningCandidates

Purpose	Provides a list of candidates who have seats allocated in the party based on
---------	--

	ranking
Prototype	public OPLVCandidate[] getWinningCandidates()
Inputs	None
Outputs	None
Called By	OPLV.assignSeats()
Calls	None
Flow of Events	<pre>OPLVCandidate[] winners = [] for i in range(0,numSeats): winners.append(candidates[i]) return winners</pre>

6. Human Interface Design

6.1 Overview of User Interface

The user will run Voting System through the terminal. In order to run the program, the user must place a correctly formatted CSV file into the same folder containing the program. On the terminal screen, the user can run the program by typing the name of the program and hitting 'Enter'. An alternative way of running the program is to type the name of the program along with the file name as the first command line argument after the program name.

After the user hits 'Enter', the program will first prompt the user to provide a file name if one wasn't provided as a command line argument. While the program is running, it will output any relevant information and notifications to the terminal screen. This includes error messages if the program closes without completing and election results if the program successfully finishes.

The audit file that is created will be output to the same folder containing the program. The user will be able to open and view the file through the folder interface.

6.2 Screen Images

Image 1. Running the program with command-line arguments

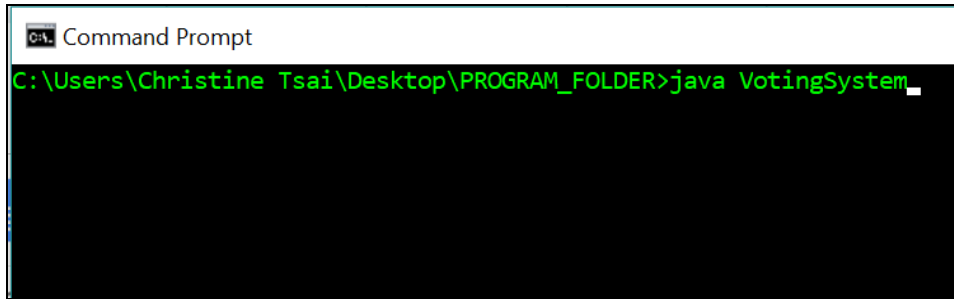


Image 2. Running the program without command-line arguments

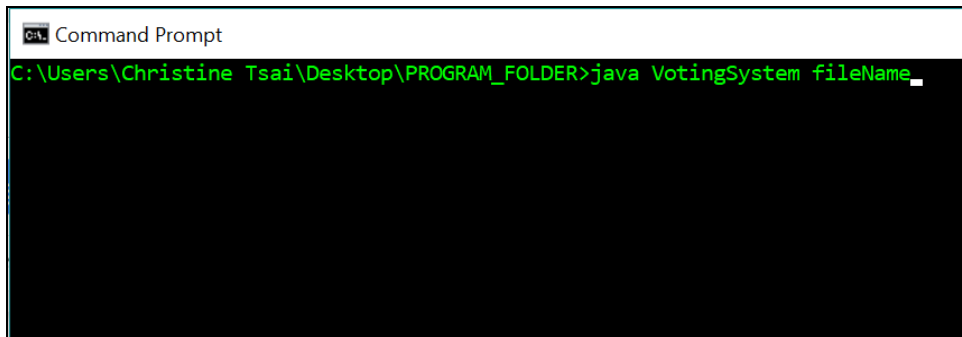


Image 3. System prompting the user to provide a file name

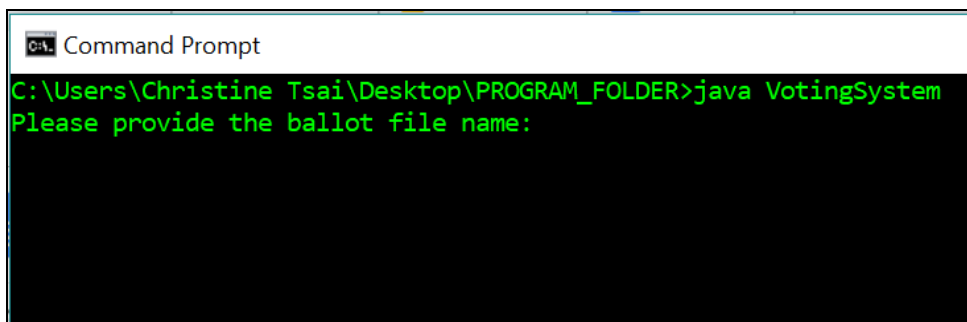
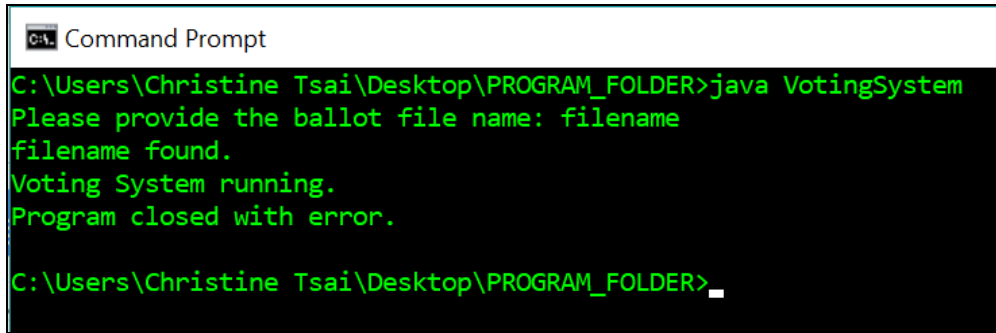


Image 4. Example of the program running, outputting feedback, and closing due to an error.



```
Command Prompt
C:\Users\Christine Tsai\Desktop\PROGRAM_FOLDER>java VotingSystem
Please provide the ballot file name: filename
filename found.
Voting System running.
Program closed with error.

C:\Users\Christine Tsai\Desktop\PROGRAM_FOLDER>
```

6.3 Screen Objects and Actions

The screen object is the terminal window which will be the environment the user will use to run the Voting System program. The user will then run the Driver class's main function to start the Voting System program in order to receive election results from the ballot file provided on the terminal.

7. Requirements Matrix

SDD class: [functions]	SRS	Functional Requirement
Driver: [main] VotingSystem: [runElection] VotingSystem: [auditResults] Audit: [Auditor]	4.1.3 Functional Requirements	SRS 4.1.3 REQ-1: Must process election results from CSV file input and create audit file.
Driver: [main]	4.1.3 Functional Requirements	SRS 4.1.3 REQ-2: Must display winner on terminal console output.