(DELETE THIS PAGE, PDF SUBMITTER!!!)

Team 19

CompuVote

Software Design Document

Names: Nikunj Chawla, Aaron Kandikatla, and Jack Fornaro

Class: CSCI 5801 Section 2

Date: (02/28/2021)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

Identify the purpose of this SDD and its intended audience. (e.g. "This software design document describes the architecture and system design of XX. ….").

This document describes the complete detailed structure of the CompuVote election system and its architecture as a part of the implementation of the requirements specified in the Software Requirements Specification (SRS) document. The document provides insights on the data structures and algorithms the CompuVote election system employs. It also assists in identifying the different systems involved in CompuVote and how they interact with each other. This Software Design Description is written for knowledgeable software professionals and architects. This includes developers, testers, and documentation writers.

1.2 Scope

Provide a description and scope of the software and explain the goals, objectives and benefits of your project. This will provide the basis for the brief description of your product.

CompuVote is a program developed in Java that is responsible for countinging votes to get election results for two voting systems: Instant Runoff Voting and Open Party List Voting. The purpose of the product is to provide a program that can tally and compute the winning candidates for the Instant Runoff and Open Party List voting systems and includes the infrastructure to easily support the addition of other voting systems. CompuVote is robust as it can process 100,000 ballots within 8 minutes. The system can be used for any situation that involves elections and requires tallying ballots compiled into the specified CSVcompiled into the specified CSV format. Businesses or corporations can also use CompuVote for their own internal voting systems. CompuVote generates an audit file that explains the entire process used to compute the winners of an election in thorough detail and generates a report file that displays a summary of the election and generates an audit file that explains the entire process used to compute the winners of an election in thorough detail and generates a report file that displays a summary of the election.

1.3 Overview

Provide an overview of this document and its organization.

- Section 2 provides an overview of the functionality, context, and design of CompuVote.
- Section 3
    - 3.1: provides a general understanding of the overall architecture of the system. It identifies the systems and the roles assigned to it. It also explains the relationships

between the modules to achieve the complete functionality of the system. Activity diagrams showing the different systems and their interactions are provided
    - 3.2: decomposes the system and explains object interactions using sequence diagrams. A UML diagram containing Java classes and their members are shown.
    - 3.3: provides a list of some of the architecture of CompuVote and explains their rationale. Some trade-offs and time issues are considered and discussed here.
- Section 4
    - 4.1 details how the data from the election files are extracted and how they are stored in the data structures.
    - 4.2 lists the system entities or major data along with their types and descriptions.
- Section 5 provides pseudocode describing what each component does.
- Section 6
    - 6.1 describes the functionality of how a user will input the CSV file into CompuVote.
    - 6.2 provides screenshots of the interface from the user's perspective.
    - 6.3 lists and discusses different actions associated with the tools that can be used to run CompuVote
- Section 7 maps use cases to functions that implement the requirements, and how the functions are actually implemented the requirements.

1.4 Reference Material

This section is optional.

List any documents, if any, which were used as sources of information for the test plan.

1.5 Definitions and Acronyms

This section is optional.

Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

2. System Overview

Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

CompuVote's main function is to successfully import and parse an election file (CSV) in either IR or OPL format and export both an audit file with the election information and a report file containing a summary of the election (the summary is also printed out to the screen at the end of an election). The functional requirements and their implementations in the design is listed in section 7. The program (which is executed through the command line) is made with the intention that it can be used as a standalone product or can also be used to integrate into another system. Some of the high level systems involved are the

operating system (user input + file retrieval), IR election process, OPL election process, and writing of files. More information about the systems and their decompositions can be found in sections 3.1 and 3.2. Regarding the data design, the input to the system is a Comma Separated Values File (CSV file). Once the file is read into the system, it must be parsed in order to extract the data. Depending on the system there will be a slightly different parsing algorithm. All the data represented in the CSV file will be saved as variables. For more information about the data design, see section 4.

3. System Architecture

3.1 Architectural Design

Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how

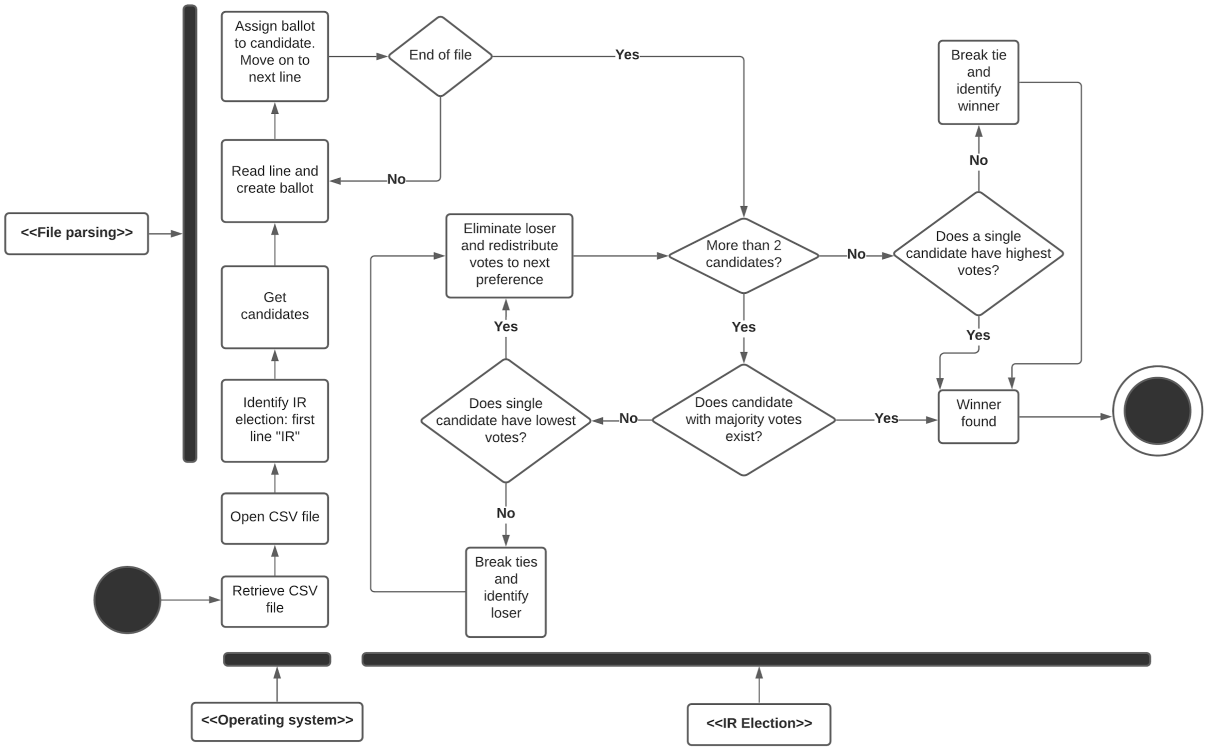
responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality. Don't go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.

High-level Subsystem	**Description and subsystem collaboration**
VotingSystemRunner	The system that will run the voting system. All methods (other than the library's) are all implemented within this subsystem.
FileInputStream	This subsystem is in charge of retrieving the input stream. The CSV file will be inputted through this stream.
FileOutputStream	This subsystem is in charge of retrieving the output streams for the audit file and the report file. Throughout the program, multitudes of information will be written to the output streams.
Class	Instances of the class Class represent interfaces in a running Java application. Every class that is instantiated (Candidate, ballot, etc.) makes use of this class.
VotingStreamParser	A subsystem of VotingSystemRunner where the CSV file is parsed for the desired voting type. Utilizes BufferedReader for parsing.
BufferedReader	The BufferedReader is used to read the lines from the CSV file, and the parsed information will fill data structures (PartyInformation, Candidate)
PrintWriter	This class implements print methods for everything within a PrintStream. This is used to print information to the output files.
StringBuilder	Allows the ability to modify string objects. Certain strings will be modified so correct election output can be produced.

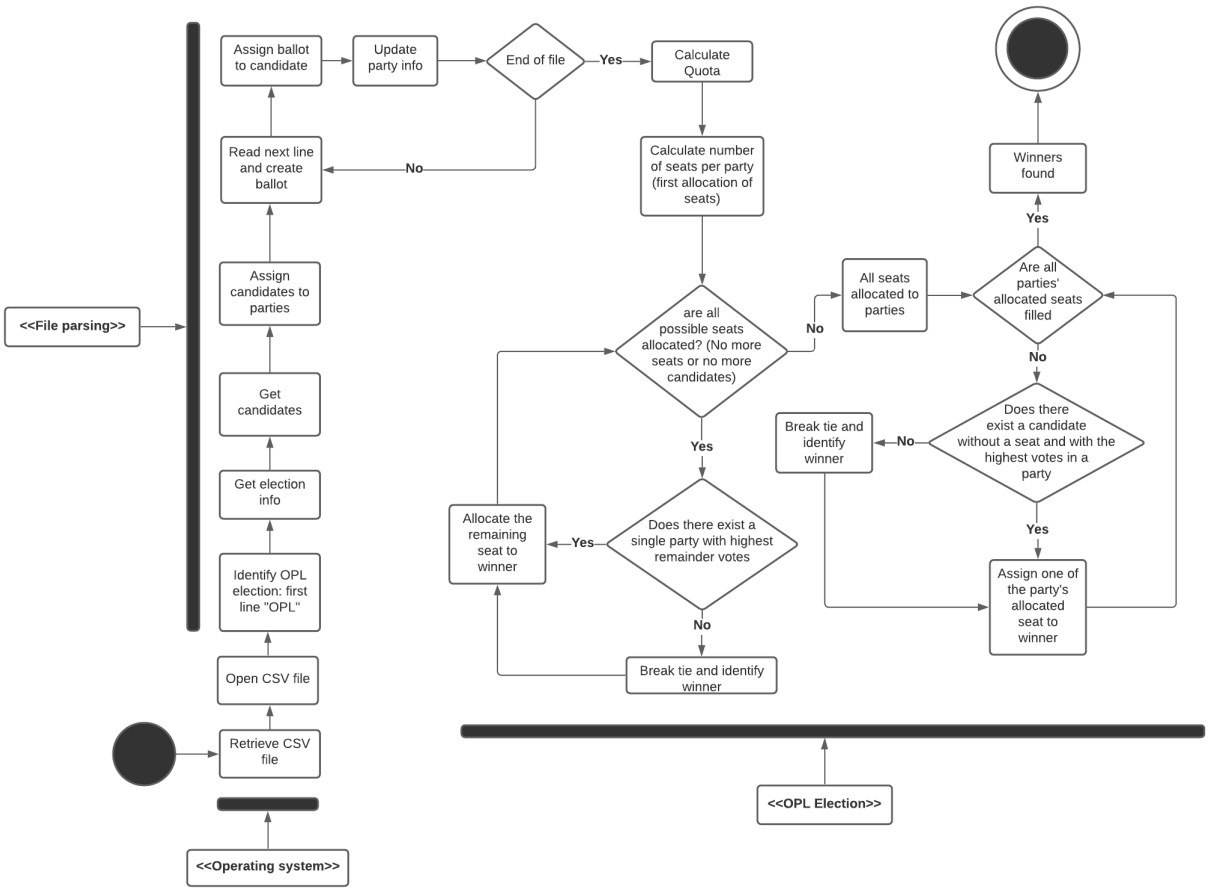
OpenPartyListingSystem	If the parsed file is determined to be of type OPL, this election system is run. This is the same level of hierarchy as InstantRunoffSystem. One of these two systems will be chosen upon reading the first line of the CSV.
InstantRunoffSystem	If the parsed file is determined to be of type IR, this election system is run. This is the same level of hierarchy as OpenPartyListingSystem. One of these two systems will be chosen upon reading the first line of the CSV.
Candidate	A class for the candidates. Will be used by PartyInformation, LinkedHashMap, and others.
Candidate[]	An array of candidates. These candidates are chosen after an OPL election when candidates are distributed their seats.
PartyInformation	A class that holds information about a given party. The number of candidates, number of seats, or number of ballots can be extracted for either election.
LinkedHashMap	A hash map that has key value pairs. The LinkedHashMap candidatesBallotMap will have keys as Candidates, and values as an ArrayDequeue of ballots. This is used to access information about a particular candidate.
Fraction	A class that allows for integer division while saving the value returned and the remainder. This class is very important in determining who gets allocated seats in an OPL election format.
Pair	The pair class holds a key and value pairing. Pairs are used throughout the code in both election formats, when a value must be stored.
ArrayDequeue	A double ended array list. This is used in the IR election format, as the candidate with the highest votes and the candidate with the lowest votes must be tracked.

ArrayList	An array that can have its size modified. ArrayLists are used to hold information about the remaining parties' seats in an OPL election format, as the number of parties can differ per election
TableFormatter	A class that assists the user in formatting a table. This will be used to format the output streams, and populate it with important election information.
Enum	A TableFormatter function, where it aligns the information to the left or the right of a table. This is used when the output streams are being formatted to a table, and the table must be populated.
Voting System	Abstract interface for both voting systems. Both voting systems will have to utilize the functions given in the interface.

Activity diagram for IR:



Activity diagram for OPL:



3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object-oriented description. For a functional description, put a top level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams,

generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.

UML Sequence Diagram Image Link:

Due to the images' large size and small text, a link to the diagram is posted below. To view the diagram through imgur, click the link below. Then click the image with your cursor (which may be a magnifying glass), and proceed to zoom in (ctrl +) and zoom out (ctrl -) as needed.

Diagram: https://i.imgur.com/e7nqq1D.jpg

UML Class Diagram Image Links:
(Due to Shana's statement that library classes must be included, the diagram must be split into two parts to be reasonably viewable. Viewing the relations diagram will clearly depict why this separation is needed.)

These images are linked externally due to being too large to properly fit in this document.

Classes: https://i.imgur.com/ZaBo39N.png
Relations: https://i.imgur.com/esMzeRD.png

3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn't choose them.

Architecture	**Rationale**
Voting System Abstract Interface	The use of this interface allows for less redundancy within the program, as it will cover occurrences that occur in both election types.
ArrayDequeue	An arrayDequeue is quite efficient in IR, as you can access the lowest candidate in a list of candidates, and the highest one. Other structures do not allow for extraction of both the first and last element.
ArrayList	ArrayLists are used here to hold information about a party's remaining seats, and the size

	of this ArrayList can change. Normal arrays do not have these capabilities.
Fraction	The fraction class allows for standardized ways of truncating and integer division. This is useful in an OPL system, as many divisions will have to take place.
FileOutputStream	The FileOutputStream allows for a uniform way to write directly to 3 different output formats, rather than saving the information and printing it to a file later. As soon as the information is calculated in the program, it will be immediately written to the output stream.
BufferedReader	Allows for a simple way to parse the file line by line. BufferedReader allows information to be read through a .get() function.
LinkedHashMap	A LinkedHashMap allows for a simple way to save key-value pairs, as well as a simple way to extract them. This is important when saving information regarding a candidate as a key, and the ballots associated as the value.

4. Data Design

4.1 Data Description

Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed and organized. List any databases or data storage items.

The candidates provided from the CSV file are stored in arrays of Candidate objects where a Candidate's main properties are its name and party.

For the IR election type, the main data structure used for tracking the ballots is a map of candidates to double-edged queues of ballots. (That is, Map<Candidate, ArrayDequeue<Ballot>> in Java). A Ballot object consists of the ballot number (based on ordering of ballots in the file), an array of Candidate objects in order of ranking, and the current index of candidate the ballot is on (incremented during each distribution).

For the OPL election type, the main two data structures used for OpenPartyListingSystem are a map of party String objects to PartyInformation objects and a map of party String objects to a map of candidates to the number of ballots they received. PartyInformation, as the name implies, contains various information corresponding to the party, including the number of candidates the party has available for seats, the number of seats available for the election, the number of ballots the party has in total, the remainder of ballots after the first allocation, and a list of Map.Entry objects with keys being the Candidate objects associated to the party and the values being the number of ballots the candidate received.

4.2 Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description in Section 3.2, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.

Type legend:

- Class field:
  - `<type>` where `<type>` is replaced with the variable's type
- Class constructor:
  - `(<params>) constructor` where `<params>` is replaced by the comma-separated list of parameter types for the constructor
- Class method:
  - `(<params>) -> <returnType>` where `<params>` is replaced by the comma-separated list of parameter types for the method and `<returnType>` is replaced by the type of the output for the method

Class	**System Entity**	**Attributes**	**Type**	**Description**
`VotingSystemRunner`	`VotingSystemRunner`	`private`	`() constructor`	A private constructor for the utility class VotingSystemRunner to prevent instantiation
`VotingSystemRunner`	`getFullFilePath`	`private, static`	`(String) -> String`	Returns the full, unique canonical form of the provided file path, exiting with a nonzero status if it cannot be resolved **Parameters:** `filePath`: The provided file path command-line argument **Returns:** The full, unique

				canonical form of the provided file path
`VotingSystemRunner`	`getFile`	`private, static`	`(String) -> FileInputStream`	Retrieves the FileInputStream for the file at the provided canonical file path, exiting with a nonzero status if it cannot be opened **Parameters:** `canonicalPath`: The canonical path from which to retrieve an input stream **Returns:** The FileInputStream for the file at the provided canonical file path **Throws:** `IOException`: Thrown if the provided file path cannot be resolved
`VotingSystemRunner`	`generateTimeStampedFileName`	`private, static`	`(String, LocalDate) -> String`	Generates a file name with a time stamp with the given prefix **Parameters:** `prefix`: The prefix for the generated file name `currentTimeStamp`: A temporal object with the current date and time **Returns:** A file name in the form `<prefix>_<year>-<month>-<day>_<hours>_<minutes>_<seconds>.txt` where the fields are replaced with the provided prefix, current date, and 24-hour time. Also, the month, day, hours, minutes, and seconds are left-padded with 0s as necessary to be 2 digits in length. **Throws:** `FileNotFoundExcep`

				`tion`: Thrown if a file cannot be found or opened from the provided file path
`VotingSystemRunner`	`main`	`public, static`	`(String[]) -> void`	Runs the election for a VotingSystem given a file path relative to the workspace **Parameters:** `args`: The command-line arguments to the program, which should only consist of one command-line argument: a path to a file, which can be absolute or relative to the current working directory.
`VotingStreamParser`	`VotingStreamParser`	`private`	`() constructor`	A private constructor for the utility class VotingStreamParser to prevent instantiation
`VotingStreamParser`	`parse`	`public, static`	`(InputStream, OutputStream, OutputStream, Map<String, Class<? extends VotingSystem>>) -> VotingSystem`	Parses a CSV election file (OPL or IR) **Parameters:** `inputStream`: The input stream of the CSV file to parse. `auditOutput`: an output stream for the audit file to write detailed information about the running of the election. `reportOutput`: An output stream for the report file to write a summary about the running of the election. `headerSystemMap`: The mapping between header strings and their corresponding VotingSystem classes **Returns:** The parsed VotingSystem **Throws:**

				`NullPointerException:`
				Thrown if any of the given streams or if the headerSystemMap is null
				
				`ParseException:`
				Thrown is there is an issue in parsing the provided InputStream

`Candidate`	`name`	`protected`	`String`	The name of the election candidate
`Candidate`	`party`	`protected`	`String`	The name of the candidate's party

`Candidate`	`Candidate`	`public`	`() constructor`	Initializes a Candidate
				
				Parameters:
				`name:`
				The name of the election candidate
				`Party:`
				The name of the candidate's party
				
				Throws:
				`NullPointerException:`
				Thrown is the provided name or party is null

`Candidate`	`getName`	`public`	`() -> String`	Returns the name of the election candidate **Returns:** The name of the election candidate of type String
`Candidate`	`getParty`	`public`	`() -> String`	Returns the name of the candidate's party **Returns:** The name of the candidate's party of type String
`Candidate`	`toString`	`public`	`() -> String`	The string form of the Candidate **Returns:** The string form of the Candidate in the form "Candidate{name=[name], party=[party]}" where [name] and [party] are the candidate's name and

				party, respectively.
`Candidate`	`equals`	`public`	`(Object) -> boolean`	Returns true if the other object is a Candidate and has the same name and party **Parameter:** `other:` The object to compare to this Candidate **Returns:** True if the other object is a Candidate and has the same name and party
`Candidate`	`hashCode`	`public`	`() -> int`	Returns the hashcode a Candidate **Returns:** The hashcode for a candidate
`Fraction`	`numerator`	`protected`	`long`	The numerator of a Fraction
`Fraction`	`denominator`	`protected`	`long`	The denominator of a Fraction
`Fraction`	`wholePart`	`protected`	`Fraction`	The floor of a Fraction
`Fraction`	`fractionalPart`	`protected`	`Fraction`	A Fraction minus its floor
`Fraction`	`reciprocalPart`	`protected`	`Fraction`	The reciprocal of a Fraction

`Fraction`	`Fraction`	`public`	`() constructor`	Initializes a Fraction given a numerator and denominator in the form of a simplified fraction
				
				Parameters:
				`numerator:`
				The numerator of the Fraction
				
				`denominator:`
				The denominator of the Fraction
				
				Throws:
				`ArithmeticException:`
				Thrown if the denominator is zero

`Fraction`	`gcd`	`protected, static`	`(long, long) -> long`	Calculates the greatest common divisor of two nonnegative numbers
				
				Parameters:
				`n1:`
				the first of two number of which to find the gcd
				
				`n2:`
				the second of two number of which to find the gcd
				
				Returns:
				The gcd of the given two nonnegative numbers

`Fraction`	`simplify`	`protected`	`() -> void`	Simplifies a Fraction by dividing the numerator and denominator by their gcf and also dividing them by -1 if the denominator is negative
`Fraction`	`getNumerator`	`public`	`() -> long`	Returns the numerator of a Fraction **Returns:** The numerator of a Fraction
`Fraction`	`getDenominator`	`public`	`() -> long`	Returns the denominator of a Fraction **Returns:** The denominator of a Fraction
`Fraction`	`getWholePart`	`public`	`() -> long`	Returns the floor of a Fraction **Returns:** The floor of a Fraction
`Fraction`	`getFractionalPart`	`public`	`() -> Fraction`	Returns a fraction minus its floor **Returns:** A fraction minus its floor
`Fraction`	`getDoubleValue`	`public`	`() -> double`	Returns the value represented by the fraction as a double **Returns:**

				The value represented by the fraction as a double
`Fraction`	`reciprocal`	`public`	`() -> Fraction`	Returns the reciprocal of a Fraction **Returns:** The reciprocal of a Fraction **Throws:** `ArithmeticException:` Thrown if the numerator is zero as the reciprocal would result in a division by zero.
`Fraction`	`add`	`public`	`(Fraction) -> Fraction`	Returns the sum of a Fraction and another Fraction **Parameters:** `other:` Another fraction to sum **Returns:** The sum of a Fraction and another fraction
`Fraction`	`subtract`	`public`	`(Fraction) -> Fraction`	Returns the difference of a Fraction and another Fraction **Parameters:** `other:` Another fraction to calculate a difference **Returns:** The difference of a Fraction and another fraction
`Fraction`	`multiply`	`public`	`(Fraction) -> Fraction`	Returns the product of a Fraction and another Fraction **Parameters:** `other:` Another fraction to calculate a product **Returns:** The product of a Fraction and another fraction

`Fraction`	`divide`	`public`	`(Fraction) -> Fraction`	Returns the quotient of a Fraction and another Fraction **Parameters:** `other:` Another fraction to calculate a quotient **Returns:** The quotient of a Fraction and another fraction
`Fraction`	`compareTo`	`public`	`(Fraction) -> int`	Returns the result of comparing two Fractions numerically **Parameters:** `other:` Another fraction to compare to **Returns:** 0 if a Fraction is equal to the other, -1 if this is less than other, and 1 if the Fraction is greater than the other
`Fraction`	`toString`	`public`	`() -> String`	Returns the string form of a Fraction **Returns:** The string form of a Fraction in the format "[numerator] / [denominator]", replacing [numerator] and [denominator] accordingly
`Fraction`	`equals`	`public`	`(Object) -> boolean`	Returns true if the other object is a Fraction and has the same numerator and denominator **Parameter:** `other:` The object to compare to a Fraction **Returns:** True if the other object is a Fraction and has the

				same numerator and denominator
Fraction	hashCode	public	() -> int	Returns the hashcode for a Fraction **Returns:** The hashcode for a Fraction
TableFormatter	intersection	public	String	The string form of the character to use in table corners or where both rows and columns intersect
TableFormatter	horizontalDivider	public	String	The string form of the character to use where two rows intersect
TableFormatter	verticalDivider	public	String	The string form of the character to use where two columns intersect
TableFormatter	TableFormatter	public	(char, char, char) constructor	Initializes a TableFormatter **Parameters:** intersection: The string form of the character to use in table corners or where both rows and columns intersect horizontalDivider: The character to use where two rows intersect verticalDivider: The character to use where two column intersect
TableFormatter	objColTableToStrRowTable	protected	(List<String>, Collection<? extends Collection<?>>, int, int) -> List<List<String>>	Given a List of string column headers and a table of objects with each List in the table representing a column, returns a table that includes the string column header and the string forms of the objects. **Parameters:** headers:

				The list of column header
				
				`colTable:`
				The headerless table of objects with each collection representing a column
				
				`numRows:`
				The number of rows in the table, including the headers as a row
				
				`numCols:`
				The number of columns in the table
				
				Returns:
				The provided table but all objects are in string form, the column headers are added, and the table is returned as a list of rows

`TableFormatter`	`getColumnLengths`	`protected`	`(List<List<String>>, int) -> int[]`	Given the string form of the table, returns the maximum lengths of the strings in each column
				
				Parameters:
				`strRowTable:`
				The string row list representation of the object column list table
				
				`numCols:`
				The number of columns the table contains
				
				Returns:
				The maximum lengths of the strings in each column of the given table

`TableFormatter`	`getTableFormat`	`protected`	`(List<Alignment>, int[], int) -> String`	Returns the formatting string for a row of data in the table
				
				Parameters:
				`alignments:`
				The list of alignments corresponding to each column
				
				`columnLengths:`

				The array consisting of the lengths of the longest strings in each column `numCols:` The number of columns in the table **Returns:** The formatting string for a row of data in the table

`TableFormatter`	`getHorizontalDivider`	`protected`	`(int[], int, String) -> String`	Returns a horizontal separator for the table
				
				Parameters:
				`columnLengths:`
				The array consisting of the lengths of the longest strings in each column
				
				`numCols:`
				The number of columns in the table
				
				`terminalCharStr:`
				The string form of the character to use on the beginning and end of the separator
				
				Returns:
				A horizontal divider for the table

`TableFormatter`	`formatAsTable`	`public`	`(List<String>, Collection<? extends Collection<?>>, List<Alignment>) -> String`	Returns a string representation of a table from a list of string column headers, a list of lists with each list representing a column, and a list of Alignment corresponding to whether a column should be left-aligned or right-aligned
				
				Parameters:
				`headers:`
				The list of column headers for the table
				
				`colTableData:`
				The headerless table of objects with each collection representing a column

				`alignments:`
				The alignments corresponding to each column
				
				Returns:
				A string representation of a table from a list of string column headers, a list of lists with each list representing a column, and a list of Alignment corresponding to whether a column should be left-aligned or right-aligned
				
				Throws:
				`NullPointerException:`
				Thrown if any of the given lists are null
				
				`IllegalArgumentException:`
				Thrown if the number of columns in provided either does not match the length of the column headers or the length of the alignments

`TableFormatter`	`toString`	`public`	`() -> String`	Returns the string form of a TableFormatter **Returns:** The string form of this TableFormatter in the form "TableFormatter{intersection=[intersection], horizontalDivider='[horizontalDivider]', verticalDivider='[verticalDivider]'}" where [intersection], [horizontalDivider], and [verticalDivider] correspond to the field variables' string forms
`TableFormatter`	`equals`	`public`	`(Object) -> bool`	Returns true if an other object is equivalent **Parameters:** `other:`

				The object to compare to a TableFormatter **Returns:** True if the other object is a TableFormatter with the same intersection, horizontalDivider, and verticalDivider
`TableFormatter`	`hashCode`	`public`	`() -> int`	Returns the hashcode for this TableFormatter **Returns:** The hashcode for this TableFormatter
`Alignment`	`LEFT`	`public, static`	`enum of Alignment`	Represents the left-alignment for a column
`Alignment`	`RIGHT`	`public, static`	`enum of Alignment`	Represents the right-alignment for a column
`Pair<K, V>`	`K`	`N/a`	`generic parameterized type of Pair`	The type corresponding to the key or first element
`Pair<K, V>`	`V`	`N/a`	`generic parameterized type of Pair`	The type corresponding to the value or second element
`Pair<K, V>`	`key`	`private`	`K`	The key or first element of a Pair
`Pair<K, V>`	`value`	`private`	`V`	The value or second element of a Pair

`Pair<K, V>`	`Pair`	`public`	`(K, V) constructor`	Initializes a Pair with the given key-value pair or pair of elements
				
				Parameters:
				`key:`
				The key of first element of a Pair
				
				`value:`
				The value or second element of a Pair

`Pair<K, V>`	`getKey`	`public`	`() -> K`	Returns the key or first element of the Pair **Returns:** The key or first element of the Pair

Pair<K, V>	getFirst	public	() -> K	Returns the key or first element of a Pair **Returns:** The key or first element of a Pair
Pair<K, V>	getValue	public	() -> V	Returns the value or second element of a Pair **Returns:** The value or second element of a Pair
Pair<K, V>	getSecond	public	() -> V	Returns the value or second element of a Pair **Returns:** The value or second element of a Pair
Pair<K, V>	toString	public	() -> String	Returns the string form of a Pair **Returns:** The string form of a Pair
Pair<K, V>	equals	public	(Object) -> boolean	Returns true if the provided object is equivalent to a Pair **Returns:** True if the provided object is equivalent to a Pair
Pair<K, V>	hashCode	public	() -> int	Returns the hashcode for a Pair **Returns:** The hashcode for a Pair

VotingSystem	VotingSystem	public	(OutputStream, OutputStream) constructor	Initializes a VotingSystem
				
				Parameters:
				`auditOutput:`
				The OutputStream to write detailed information about the running of the election
				
				`reportOutput:`
				The OutputStream to write a summary about the running of the election

				Throws: `NullPointerException:` Thrown if either auditOutput or reportOutput is null
`VotingSystem`	`getCandidateHeaderSize`	`public, abstract`	`() -> int`	Returns the number of lines that makes up the header for the candidates **Returns:** The number of lines that makes up the header for the candidates
`VotingSystem`	`getBallotHeaderSize`	`public, abstract`	`() -> int`	Returns the number of lines that makes up the header for the ballots **Returns:** The number of lines that makes up the header for the ballots

`VotingSystem`	`importCandidatesHeader`	`public, abstract`	`(String[], int) -> void`	Parses the lines corresponding to the header for the candidates
				
				Parameters:
				`header:`
				The lines corresponding to the header
				
				`line:`
				The line number associated with the first line of the header
				
				Throws:
				`ParseException:`
				Thrown if there is an issue in parsing the header

`VotingSystem`	`addCandidates`	`public, abstract`	`(String, int) -> void`	Parses a String corresponding to candidates and party and adds them internally
				
				Parameters:
				`candidates:`
				The String representing the list of candidates and their parties
				
				`line:`

				The line number associated with the candidates String **Throws:** `ParseException:` Thrown if there is an issue in parsing the header

`VotingSystem`	`importBallotsHeader`	`public, abstract`	`(String[], int) -> void`	Parses the lines corresponding to the header for the ballots
				
				Parameters:
				`header:`
				The lines corresponding to the header
				
				`line:`
				The line number associated with the first line of the header
				
				Throws:
				`ParseException:`
				Thrown if there is an issue in parsing the header

`VotingSystem`	`addBallot`	`public, abstract`	`(String, int) -> void`	Parses a line corresponding to a ballot and adds it internally
				
				Parameters:
				`ballotNumber:`
				The number corresponding to the current ballot
				
				`ballot:`
				The String corresponding to a ballot
				
				`line:`
				The line number associated with the current ballot line being read
				
				Throws:
				`ParseException:`
				Thrown if there is an issue in parsing the current ballot

`VotingSystem`	`getName`	`public, abstract`	`() -> String`	Returns the name of a voting system

				Returns: The name of a voting system
VotingSystem	getShortName	public, abstract	() -> String	Returns the short name for a voting system; that is, the name that appears at the top of an election file **Returns:** The short name for the voting system
VotingSystem	getNumCandidates	public, abstract	() -> int	Returns the number of candidates that a VotingSystem contains **Returns:** The number of candidates that a VotingSystem contains
VotingSystem	getCandidates	public, abstract	() -> Collection<Candidate>	Returns a collection of Candidates involved in the election **Returns:** A collection of Candidates involved in the election
VotingSystem	getNumBallots	public, abstract	() -> int	Returns the number of ballots that a VotingSystem contains **Returns:** The number of ballots that a VotingSystem contains
VotingSystem	runElection	public, abstract	() -> void	Runs the election for the VotingSystem and determines the winner
VotingSystem	toString	public, abstract	() -> String	Returns the string form of a VotingSystem **Returns:** The string form of a VotingSystem
Ballot	ballotNumber	protected	int	The ballot number associated with this ballot, created from 1 to the number of ballots in

				order of the ballots in the file
Ballot	candidateIndex	protected	int	The index corresponding to the current candidate this ballot is on at the current stage of eliminations
Ballot	rankedCandidates	protected	Candidate[]	The array of candidates that the ballot ranked, in order of ranking
Ballot	Ballot	protected	(Candidate[], int) constructor	Initializes a Ballot **Parameters:** rankedCandidates: The array of candidates that the ballot ranked, in order of ranking ballotNumber: The index corresponding to the current candidate this ballot is on at the current stage of eliminations
Ballot	getNextCandidate	protected	() -> Candidate	Returns the next ranked Candidate for a ballot **Returns:** The next ranked Candidate for a ballot
Ballot	toString	public	() -> String	Returns the String form of a Ballot **Returns:** "Ballot <ballotNumber>: <rankedCandidates>" where <ballotNumber> and <rankedCandidates> are replaced with string representations of their respective fields
Ballot	equals	public	(Object) -> boolean	Returns true if the provided object is equivalent to the Ballot **Returns:** True if the provided object is equivalent to the Ballot
Ballot	hashCode	public	() -> int	Returns the hashcode for a Ballot

				Returns: The hashcode for a Ballot
InstantRunoffSystem	numCandidates	protected	int	The number of candidates in an IR election
InstantRunoffSystem	numBallots	protected	int	The total number of Ballots
InstantRunoffSystem	candidates	protected	Candidate[]	The candidates for this election in the order provided
InstantRunoffSystem	candidateBallotsMap	protected	Map<Candidate, Deque<Ballot>>	A mapping of Candidate objects to an ArrayDeque of the Ballot objects currently belonging to them
InstantRunoffSystem	auditOutput	protected	OutputStream	An output stream for the audit file to write detailed information about the running of the election.
InstantRunoffSystem	reportOutput	protected	OutputStream	An output stream for the report file to write a summary about the running of the election.

InstantRunoffSystem	InstantRunoffSystem	public	(OutputStream, OutputStream) constructor	Initializes an InstantRunoffSystem
				
				Parameters:
				`auditOutput:`
				An output stream for the audit file to write detailed information about the running of the election.
				
				`reportOutput:`
				An output stream for the report file to write a summary about the running of the election.

InstantRunoffSystem	getCandidateHeaderSize	public	() -> int	Returns the size of the candidate header, which is equal to 1 for IR **Returns:** The size of the candidate header, which is equal to 1 for IR

`InstantRunoffSystem`	`getBallotHeaderSize`	`public`	`() -> int`	Returns the size of the ballot header, which is equal to 1 for IR **Returns:** The size of the ballot header, which is equal to 1 for IR

`InstantRunoffSystem`	`importCandidatesHeader`	`public`	`(String[], int) -> void`	Parses and stores the number of candidates
				
				Parameters:
				`header:`
				An Array of String containing candidate header information from CSV file
				
				`line:`
				The line number in the CSV file

`InstantRunoffSystem`	`addCandidates`	`public`	`(String, int) -> void`	Parses and adds candidates if candidates matches the form of comma-separated groups of "<candidate> (<party>)" where <candidate> and <party> are nonempty strings
				
				Parameters:
				`candidates:`
				String of candidates and their parties
				
				`line:`
				The line number in the CSV file

`InstantRunoffSystem`	`importBallotsHeader`	`public`	`(String[], int) -> void`	Parses and stores ballot header information
				
				Parameters:
				`header:`
				An Array of String containing ballot header information from CSV file
				
				`line:`
				The line number in the CSV file

`InstantRunoffSystem`	`addBallot`	`public`	`(String, int) -> void`	Parses and adds ballots if ballot consists comma-separated values

				of 1 through m, iterating by 1, where m <= numCandidates, with empty strings filling up the remaining numCandidates - m spaces
				
				Parameters:
				`ballotNumber:`
				The ballot number associated with the ballot
				
				`ballot:`
				The String containing the unparsed ballot from the CSV file
				
				`line:`
				The line number in the CSV file

`InstantRunoffSystem`	`getName`	`public`	`() -> String`	Returns the name of the election type - “Instant Runoff Voting” **Returns:** The name of the election type - “Instant Runoff Voting”
`InstantRunoffSystem`	`getShortName`	`public`	`() -> String`	Returns the short name of the election type - “IR” **Returns:** The short name of the election type - “IR”
`InstantRunoffSystem`	`getNumCandidates`	`public`	`() -> int`	Returns the number of candidates **Returns:** The number of candidates
`InstantRunoffSystem`	`getCandidates`	`public`	`() -> Collection<Candidate>`	Returns a collection of all Candidates **Returns:** A collection of all Candidates
`InstantRunoffSystem`	`getNumBallots`	`public`	`() -> int`	Returns the number of ballots **Returns:** The number of ballots

InstantRunoffSystem	toString	public	() -> String	Returns the String form of InstantRunoffSystem - “InstantRunoffSystem” **Returns:** The String form of InstantRunoffSystem - “InstantRunoffSystem”
InstantRunoffSystem	getLowestHigheestCandidates	protected	() -> Pair<Pair<Integer, ArrayList<Candidate>>, Pair<Integer, Candidate>>	Returns the candidate with the highest votes and the candidate with the lowest votes **Returns:** the candidate with the highest votes and the candidate with the lowest votes in the structure of Pair<Pair<Integer, ArrayList<Candidate>>, Pair<Integer, Candidate>>
InstantRunoffSystem	eliminateLowest	protected	(Candidate) -> void	Eliminates a candidate and redistributes their ballots **Parameters:** lowest: The candidate who is eliminated and need’s their ballots redistributed
InstantRunoffSystem	runElection	public	() -> void	Runs the IR election algorithm
PartyInformation	numCandidates	protected	int	Number of candidates in a party
PartyInformation	numSeats	protected	int	Number of seats given to a party
PartyInformation	numBallots	protected	int	Number of ballots a party received
PartyInformation	remainder	protected	Fraction	A fraction object storing the remaining votes after initial allocation in OPL
PartyInformation	orderedCandidateBallots	protected	List<Map.Entry<Candidate, Integer>>	The pairs of candidates for this party and the number of ballots they were given, sorted by number of ballots the candidate was given.

PartyInformation	PartyInformation	protected	() constructor	Initializes PartyInformation
PartyInformation	toString	public	() -> String	Returns the String form of PartyInformation **Returns:** The String form of PartyInformation
OpenPartyListSystem	numCandidates	protected	int	The number of Candidates in the OPL election **Returns:** The number of candidates
OpenPartyListSystem	numBallots	protected	int	The total number of ballots in the OPL election **Returns:** The total number of ballots
OpenPartyListSystem	numSeats	protected	int	The total number of seats in the OPL election **Returns:** The total number of seats
OpenPartyListSystem	candidates	protected	Candidate[]	The array of Candidate objects for OpenPartyListSystem in the order provided
OpenPartyListSystem	partyToCandidateCounts	protected	Map<String, Map<Candidate, Integer>>	A mapping of parties to a another mapping of their candidates and ballot counts
OpenPartyListSystem	partiesToPartyInformation	protected	Map<String, PartyInformation>	A mapping of parties to their respective PartyInformation instance
OpenPartyListSystem	auditOutput	protected	OutputStream	An output stream for the audit file to write detailed information about the running of the election.
OpenPartyListSystem	reportOutput	protected	OutputStream	An output stream for the report file to write a summary about the running of the election.

OpenPartyListSystem	tableFormatter	protected	TableFormatter	The TableFormatter used to create text tables
OpenPartyListSystem	OpenPartyListSystem	public	(OutputStream, OutputStream) constructor	Initializes OpenPartyListSystem **Parameters:** auditOutput: An output stream for the audit file to write detailed information about the running of the election. reportOutput: An output stream for the report file to write a summary about the running of the election.
OpenPartyListSystem	getCandidateHeaderSize	public	() -> int	Returns the size of candidate header for OPL in CSV file: 1 **Returns:** The size of candidate header for OPL in CSV file: 1
OpenPartyListSystem	getBallotHeaderSize	public	() -> int	Returns the size of ballot header for OPL in CSV file: 2 **Returns:** The size of ballot header for OPL in CSV file: 1
OpenPartyListSystem	importCandidatesHeader	public	(String[], int) -> void	Parses the header information for candidates **Parameters:** header: The array of Strings corresponding to the lines of the CSV file for the candidate header line: Line number in the CSV file
OpenPartyListSystem	addCandidates	public	(String, int) -> void	Creates Candidates and Parties candidates matches the form of comma-separated groups of "[<candidate>, <party>]" where

				<candidate> and <party> are nonempty strings
				
				Parameters:
				`candidates:`
				Unparsed String from the CSV file containing candidates and their parties
				
				`line:`
				Line number in the CSV file

`OpenPartyListSystem`	`importBallotsHeader`	`public`	`(String[], int) -> void`	Reads ballot header information
				
				Parameters:
				`header:`
				The array of Strings corresponding to the lines of the CSV file for the ballots header
				
				`line:`
				The line number in the CSV file

`OpenPartyListSystem`	`addBallot`	`public`	`(String, int) -> void`	Parses and adds a ballot for a candidate and party if the ballot consists of numCandidates comma-separated values with one of them being 1 and the rest being empty
				
				Parameters:
				`ballotNumber:`
				The ballot number associated with the ballot
				
				`ballot:`
				The String containing the unparsed ballot from the CSV file
				
				`line:`
				The line number in the CSV file

`OpenPartyListSystem`	`getName`	`public`	`() -> String`	Returns the name for OpenPartyListSystem - “Open Party List Voting” **Returns:**

				The name for OpenPartyListSystem - “Open Party List Voting”
`OpenPartyListSystem`	`getShortName`	`public`	`() -> String`	Returns the short name for OpenPartyListSystem - “OPL” **Returns:** The short name for OpenPartyListSystem - “OPL”
`OpenPartyListSystem`	`getNumCandidates`	`public`	`() -> int`	Returns the number of candidates **Returns:** The number of candidates
`OpenPartyListSystem`	`getCandidates`	`public`	`() -> Collection<Candidate>`	Returns a collection of Candidates in the election **Returns:** A collection of candidates in the election
`OpenPartyListSystem`	`getNumBallots`	`public`	`() -> int`	Returns the number of ballots **Returns:** The number of ballots
`OpenPartyListSystem`	`toString`	`public`	`() -> String`	Returns the String form of OpenPartyListSystem **Returns:** The String form of OpenPartyListSystem - “OpenPartyListSystem”
`OpenPartyListSystem`	`allocateInitialSeats`	`protected`	`(Fraction) -> Pair<Integer, Set<String>>`	Allocates initial seats based off the quota **Parameters:** `quota:` The total number of votes / seats to calculate the initial seat allocations per party **Returns:** A pair of number of seats remaining after initial allocation and set of String representing

				parties able to receive additional seats
OpenPartyListSystem	indexAfterEquivalentGroup	protected	<T> (List<T>, int, Comparator<T>) -> Pair<Integer, ArrayList<T>>	Given a sorted ArrayList, an index from which to begin, and a comparator for comparing elements in the ArrayList, returns the next group of equivalent elements as determined by the comparator in addition to the index after the last added element. **Parameters:** arrayList: The sorted ArrayList from which to retrieve the next equivalent ordered group idx: The index from which to begin retrieving the group comparator: The comparator used to compare elements when determining if elements are equivalent and should be added to the group **Returns:** An ArrayList of equivalent elements in the provided ArrayList as determined by the provided comparator, starting at the provided index.
OpenPartyListSystem	allocateRemainingSeats	protected	(int, Set<String>) -> void	Allocates remaining seats to parties with the highest remaining votes **Parameters:** numSeatsRemaining: Number of seats remaining after initial allocation remainingParties: Parties that still have enough candidates for additional seats

`OpenPartyListSystem`	`distributeSeatsToCandidates`	`protected`	`() -> void`	Distributes each party's seats to their candidates by popularity

5. Component Design

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

Pseudocode: https://imgur.com/a/lxLjTa5

See 4.2 for textual descriptions of the class fields, constructors, and methods displayed in the pseudocode.

6. Human Interface Design

6.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

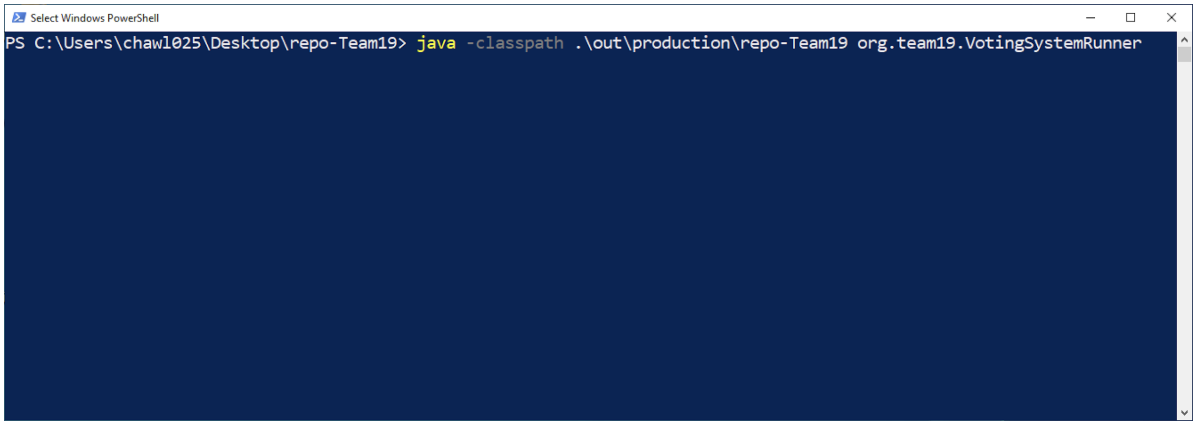
The user, who is person compiling and inputting the CSV file, will be doing so through CLI (command line interface) through the appropriate operating system (explanation for usage per OS in section 3.3 of the Software Requirements Specification Document). The file path to the election file is the only command-line argument required for the program to run. In the repo-Team19 directory, one runs the command `java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>` to run the program where <pathToElectionFile> is replaced with the file path to the election file.

6.2 Screen Images

Display screenshots showing the interface from the user's perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

macOS Terminal:

Windows PowerShell:



Ubuntu gnome-terminal:

6.3 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

CLI:

- *Exit*: Closes Terminal.
- *Maximize/Store Down*: Maximize the size of the window/restore to original size.
- *Minimize*: Temporarily closes the terminal window, but can be reopened in the same state.
- *Scroll Bar*: A widget that allows one to scroll in a predetermined direction.

Ubuntu CLI:

- File
    - *New Tab*: Opens a new tab within the current terminal window.
    - *New Window*: Opens a new terminal window.
    - *Close Tab*: Closes the current tab.
    - *Close Window*: Closes the current window.
- Edit
    - *Copy*: Makes a copy of the current highlighted message.

    - *Copy as HTML*: Makes a copy of the current highlighted message in HTML format.
    - *Paste*: Paste the current copied contents.
    - *Select All*: Highlights all contents in the terminal.
    - *Preferences*: Changes the terminal preferences for a given profile.
- View
    - *Show Menubar*: Checkbox that will determine if the menubar (bar with File, Edit, View, Search, Terminal, and Help) is displayed.
    - *Full Screen*: Maximizes the window to cover the entire screen.
    - *Zoom In*: Increases text size.
    - *Normal Size*: Returns to 100% zoom.
    - *Zoom Out*: Decreases text size.
- Search
    - *Find…*: Opens a search box to find a given word/phrase.
    - *Find Next*: Finds the next instance of a given word/phrase.
    - *Find Previous*: Finds the previous instance of a given word/phrase.
    - *Clear Highlight*: Clears currently highlighted text
- Terminal
    - *Read-Only*: Sets the mode to Read-Only. The user cannot input anything directly into the command line.
    - *Reset*: Fixes previous errors after escape sequences.
    - *Reset and Clear*: Fixes previous errors after escape sequences and clears the entire terminal of text.
- Help
    - *Contents*: Opens a window about how to work with Terminal. Gives a general introduction, as well as hyperlinks to other helpful features.
    - *About*: Opens a window that gives the information about GNOME Terminal, such as the current version, copyright, and a link to the website.

7. Requirements Matrix

Provide a cross reference that traces components and data structures to the requirements in your SRS document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

Function ID	**Existence in Pseudocode**	**Explanation of Implementation**
UC_001	`args`	The actor inputs the CSV file path into their respective CLI (based on OS).
UC_002	`In public static void main(String[]`	In the public static void main(), the inputStream

	`args):`
	
	`try:`
	
	`inputStream = retrieveInputStream(args[0])`
	`except:`
	
	`presentError()`
		checks that the first argument is the correct file stream. If it is not the correct file path, then an error will appear. Otherwise, CompuVote will be run.

UC_003	`public class VotingStreamParser:`
	
	`public VotingSystem parse(InputStream file, OutputStream auditFile, OutputStream reportFile,...`
		This class will parse the election file for the information required to simulate an election. It will be parsed into 1 of 2 formats, one for IR, and one for OPL.

UC_004	`In public class VotingStreamParser`
	
	`lineNumber = 1`
	
	`try:`
	`firstLine = file.getLine()`
	`except:`
	
	`presentError(lineNumber)`
	
	`votingSystemClass = headerSystemMap.get(firstLine)`
		The system will identify the type of voting system, and the file will be parsed based on that election type. The election type will be determined. Otherwise, the program will exit with nonzero status.

UC_005	`In public class VotingStreamParser`
	
	`candidateHeaderSize = votingSystem.getCandidateHeaderSize()`
	`try:`
	
	`candidatesHeader =`
		The system will parse the election file for the number of candidates, and for the list of candidates and their corresponding parties in the IR election type.

	file.getLines(candidateHeaderSize)
	except:
	
	presentError(lineNumber)
	try:
	
	votingSystem.importCandidatesHeader(candidatesHeader, lineNumber)
	except:
	
	presentError(lineNumber)
	
	try:
	candidatesLine = file.getLine()
	except:
	
	presentError(lineNumber)
	try:
	
	votingSystem.addCandidates(candidatesLine)
	except:
	
	presentError(lineNumber)
	if len(votingSystem.getCandidates()) != numCandidates:
	
	presentError(lineNumber)	

UC_006	In public class VotingStreamParser ballotHeaderSize = votingSystem.getBallotHeaderSize() try:	The system must parse the election file for the ballot information in an IR election type. The number of ballots will be returned, as well as a ballot for each line where a ballot must be parsed.

	``ballotHeader = file.getLines(ballotHeaderSize)
	  except presentError(lineNumber)
	  try:
	
	votingSystem.importBallotHeader(ballotHeader, lineNumber)
	  except:
	
	presentError(lineNumber)
	
	foreach remaining line:
	  try:
	
	votingSystem.addBallot(line, lineNumber, ballotNumber)
	  except:
	
	presentError(lineNumber)
	
	lineNumber++
	
	ballotNumber++```	

UC_007	```In public class VotingStreamParser
	
	candidateHeaderSize = votingSystem.getCandidateHeaderSize()
	  try:
	
	candidatesHeader = file.getLines(candidateHeaderSize)
	  except:
	
	presentError(lineNumber)```
	
		The system will parse the election file for the number of candidates, and for the list of candidates and their corresponding parties in the OPL election type

	try:
	
	votingSystem.importCandidatesHeader(candidatesHeader, lineNumber)
	except:
	
	presentError(lineNumber)
	
	try:
	candidatesLine = file.getLine()
	except:
	
	presentError(lineNumber)
	try:
	
	votingSystem.addCandidates(candidatesLine)
	except:
	
	presentError(lineNumber)
	if len(votingSystem.getCandidates()) != numCandidates:
	
	presentError(lineNumber)	

UC_008	In public class VotingStreamParser ballotHeaderSize = votingSystem.getBallotHeaderSize() try: ballotHeader = file.getLines(ballotHeaderSize) except presentError(lineNumber)	The system must parse the number of seats for the OPL election type.

	`try:`
	
	`votingSystem.importBallotHeader(ballotHeader, lineNumber)`
	`except:`
	
	`presentError(lineNumber)`	

UC_009	`In public class VotingStreamParser`
	
	`foreach remaining line:`
	`try:`
	
	`votingSystem.addBallot(line, lineNumber, ballotNumber)`
	`except:`
	
	`presentError(lineNumber)`
	
	`lineNumber++`
	
	`ballotNumber++`
	
	`if numBallots != ballotNumber - 1:`
	
	`presentError(lineNumber - 1)`
		The system must parse the election file for the ballot information in an IR election type. The number of ballots will be returned, as well as a ballot for each line where a ballot must be parsed.

UC_010	`public class InstantRunoffSystem extends VotingSystem:` `public void runElection():`	This class will run the voting system in an IR format. A candidate will be declared the winner.
UC_011	`public class InstantRunoffSystem extends Voting System:` `public void runElection():`	The voting system will identify the candidate(s) with the lowest ballot count and the one(s) with the highest. If a candidate has more than 50% of the votes, they are

	`if candidateBallotsMap.length() <= 2:`
	`  …`
	`else`
	`  ...`
		declared the winner.

UC_012	`protected void eliminateLowest(Candidate lowest):`	The candidate with the lowest votes is eliminated. This candidate will be removed from the candidateBallotsMap().

UC_013	`public class InstantRunoffSystem extends Voting System:`
	`  public void runElection():`
	
	`  if candidateBallotsMap.length() <= 2:`
	`  int firstSecondCandidateComparison = candidateBallotsMap.getFirst().compare(candidateBallotsMap.getSecond()),`
	`  if firstSecondCandidateComparison > 0:`
	`  print firstCandidate`
	`  else if firstSecondCandidateComparison < 0:`
	`  print secondCandidate`
	`  else`
	`  ...`
		If the system determines no candidate has a majority and all eliminations have taken place, then the candidate with the most votes will win. The victor will be printed.

UC_014	`In protected void eliminateLowest(Candidate lowest):`
	
	`  for ballot in ballotsToRedistribut`
		Once the lowest candidate is determined, their ballots will be redistributed do the candidateBallotsMap() based on their secondary preferences.

	e: Candidate nextCandidate = ballot.getNextCandidate() if nextCandidate != UNDEFINED: candidateBallotsMap.get(nextCandidate).add(ballot)	

UC_015	public class InstantRunoffSystem extends Voting System:
	public void runElection():
	
	if candidateBallotsMap.length() <= 2:
	…
	else
	print randomlySelect(firstCandidate, secondCandidate)
		In the case that 2 or more candidates have the same number of votes, randomlySelect randomly selects a candidate to be the winner. That candidate is printed.

UC_016	public class InstantRunoffSystem extends Voting System: public void runElection(): Candidate winner int firstSecondCandidateComparison = candidateBallotsMap.getFirst().compare(candidateBallotsMap.getSecond()), if firstSecondCandidateComparison > 0: print	The winner is declared, and their information will be printed.

	firstCandidate
	else if
	firstSecondCandidate
	Comparison < 0:
	  print
	secondCandidate
	else:
	  print
	randomlySelect(first
	Candidate,
	secondCandidate)	

UC_017	public class OpenPartyListingSyst em extends VotingSystem: void runElection():	This class will run the voting system in an OPL format. All unfilled seats will be declared.
UC_018	In public class OpenPartyListingSyst em extends VotingSystem: public void addCandidates(String candidates, int line):	In the function addCandidates, all the independents will be added to their respective party's DS.
UC_019	In public class OpenPartyListingSyst em extends VotingSystem: public void addCandidates(String candidates, int line):	In the function addCandidates, all the remaining candidates will be added to their respective party's DS.
UC_020	In public class OpenPartyListingSyst em extends VotingSystem: void runElection(): Fraction quota = Fraction(numBallots, numSeats)	The quota will be calculated through integer division while utilizing an instance of the Fraction class.
UC_021	In public class OpenPartyListingSyst	The function allocateInitialSeats will

	em extends VotingSystem: int allocateInitialSeats (Fraction quota):	calculate for the number of seats each party is allocated, and also saves the remaining number of votes truncated from each party.
UC_022	In public class OpenPartyListingSystem extends VotingSystem: void allocateRemainingSeats():	The function allocateRemainingSeats will allocate the remaining seats (if any) by comparing each party's remaining votes.
UC_023	In public class OpenPartyListingSystem extends VotingSystem: UC_022: void allocateRemainingSeats() chosenParty = randomlySelect(highestRemainingParties) UC_025: ArrayDequeue<Candidate> distributeSeatsToCandidates(): Candidate selected = selectRandom(highestRemainingCandidates)	This function will break a tie in UC_022 or UC_025. For UC_022, the chosenParty will be randomly selected from the highestRemainingParties. For UC_025, the selected Candidate will be randomly selected from the highestRemainingCandidates.
UC_024	In public class OpenPartyListingSystem extends VotingSystem: ArrayDequeue<Candidate> distributeSeatsToCandidates():	All seats are declared after the function is run.
UC_025	In public class OpenPartyListingSyst	All the seats are allocated for each seat in the party. This

	em extends VotingSystem: ArrayDequeue<Candidate> distributeSeatsToCandidates():	process continues until all seats are distributed to all parties through distributeSeatsToCandidates() .
UC_026	public String formatAsTable(List<String> headers, List<List<Object>> colTableData, List<Alignments> alignments):	This function will format the table for all output files, as well as the general output format for displaying the summary to the screen immediately after completion. The summary will have the same information that the report file will have.
UC_027	this.auditOutput = auditOutput	This instantiates the outputStream for the audit file. Throughout the program, information is outputted to the output stream for the audit file.
UC_028	this.reportOutput = reportOutput	This instantiates the outputStream for the report file. Throughout the program, information is outputted to the output stream for the report file.

8. APPENDICES

This section is optional.

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.