Use Case Document

Prepared by Team 08 members: Jashwin Acharya (achar061), Steve Petzold (petzo017), Vincent Hoang (hoang317), and Carlos Chasi-Mejia (chasi009)

Use Case #1: Command Line Interface

Name:	Command Line Interface
ID:	1
Description:	The user should be able to enter the name of the CSV file containing the ballots and voting protocol information.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Voting calculation can be automated by simply entering the input CSV file name.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	The user runs the Java program using the appropriate commands and is prompted to enter the name of the CSV file.
Pre-conditions:	CSV file should be present in the project directory and should be readable.
Post-conditions:	Program accepts the file name if it is correct and proceeds to parse the header of the file first before parsing Candidate and Ballots information.
Main Course:	1. User runs program 2. Command Line prompts user to enter file name 3. If correct file name is entered, then process it 4. If in-correct file name is entered, then re-prompt user
Alternate Courses:	If the wrong file name is entered, then the user is re-prompted infinitely many times until they enter the right file name.
Exceptions:	Wrong File Name entered: 1. If the wrong file name is entered, then the user is re-prompted to enter the correct file name.

Use Case #2: Parse CSV file header.

Name:	Parse CSV file header
ID:	2
Description:	Process CSV header file to identify the voting protocol to use for the election.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Voting protocol is identified programmatically without the user having to enter the voting protocol name explicitly whenever they run the program.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	The user enters the correct CSV file name when prompted by the command line.
Pre-conditions:	The CSV file always has the same format with the header being the first line followed by Candidate and Ballot information.
Post-conditions:	Program determines the correct voting protocol to be used and proceeds to process the number of ballots to determine a winner.
Main Course:	1. User runs Java program. 2. User enters the correct CSV file name. 3. Header is parsed. 4. A message is displayed in the Command Line Interface that the header was successfully parsed.
Alternate Courses:	No alternate course needed as the file format is always the same.
Exceptions:	File Corrupted 1. Inform user that a read error occurred and prompt them to enter the file name again

Use Case #3: Parse Ballots and Candidate Information

Name:	Parse Ballots and Candidate Information
ID:	3
Description:	Parse each line of the CSV file after the header is processed to determine the number of candidates, their information, how many ballots they received and optionally the number of seats

	(only applicable for Open Party List Voting).
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Ballot calculations are automated without users having to calculate them manually.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	The user enters the CSV file name and the header is done being parsed.
Pre-conditions:	1. CSV file is present in the same directory as project files. 2. CLI is implemented and allows users to enter CSV file name. 3. The file format is consistent with the following format: a. First line has the voting protocol name. b. Second line specifies the number of candidates. c. Third line specifies candidate names and party information. d. Fourth line specifies the number of seats if the voting protocol is Open Party List or it specifies the number of ballots e. If the voting protocol is Open Party List, then the fifth line specifies the number of ballots. 4. The file header is successfully processed without any errors.
Post-conditions:	Ballot and candidate information is successfully processed and stored in a data structure within our Java program.
Main Course:	1. Once header is processed, the second line specifying the number of candidates is processed. 2. The third line of the file specifying candidate names and party affiliations is processed. 3. The fourth line containing the number of seats (for OPL) or number of ballots is processed. 4. If needed, the 5th line is processed for the number of ballots if voting protocol is OPL. 5. Program proceeds to calculate ballots.
Alternate Courses:	No alternate course needed as the CSV file always has the same format.
Exceptions:	File Corrupted 1. Inform user that a read error occurred and prompt them to enter the file name again

Use Case #4: Produce Audit File

Name:	Produce Audit File
ID:	4
Description:	The system needs to produce an audit file that shows the progress of the election and contains all election information such as voting protocol name, number of candidates, ballot information, winner information, etc.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	The audit file provides a transparent record of how the voting progressed and can be used to detect and fix any errors that might have occurred in the vote calculation process.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	The audit file is produced once the ballot calculations are done and a winner is decided.
Pre-conditions:	1. CSV file is present in the same directory as project files 2. CLI allows user to input CSV file name. 3. CSV File is processed successfully including headers and ballot information. 4. Ballot calculations are performed using the appropriate voting protocol specified in the file header. 5. Audit file will be produced in the same directory as the project files. 6. Order of ballots assigned to candidates must be maintained. 7. Audit file should show the order of candidates removed in the IR voting process 8. Audit file should show the order of ballots assigned to candidates after the lowest voted candidate is removed
Post-conditions:	1. Audit file is produced containing an accurate step-by-step record of how the election progressed.
Main Course:	1. Audit file is created and contains the timestamp in its filename to maintain uniqueness. 2. All election information including Type of Voting, Number of Candidates, Candidates, Number of Ballots, calculations, etc will be written to the audit file. 3. Once all the information is written, the audit file connection can be closed.
Alternate Courses:	One possible issue could be duplicate file names when trying to create an audit file, but we circumvent that by attaching a

	timestamp to each audit file name to ensure uniqueness.
Exceptions:	Hardware Failure 1. If the system undergoes a hardware failure before the audit file is produced, then the user would have to run the program again. File Write Error: 1. In case of a file write error the user would have to run the program again from scratch.

Use Case #5: Majority wins when using IR Voting Protocol

Name:	Majority wins when using IR Voting Protocol
ID:	5
Description:	The IR voting protocol must determine a winner if they receive a majority (greater than 50% of the total votes)
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Ballot calculations are automated and manual labor is reduced.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	Ballots and Candidate information is successfully parsed from the CSV file.
Pre-conditions:	1. CSV file is present in the same directory as project 2. CLI is implemented to allow user to enter file name 3. File format is always the same: a. 1st line of file is the voting protocol, which in this case is “IR”. b. 2nd line of the file contains the number of candidates. c. 3rd line of the file contains the candidates and their party separated by commas. d. 4th line contains the number of ballots in the file. e. The rest of the file contains information regarding which candidates were chosen by the voters. 4. There are no errors in the ballots. 5. Each ballot will have at least 1 ranking. 6. The ranking numbers will not have any issues. 7. No write-in candidates. 8. A voter can rank from 1 to the number of candidates.

Post-conditions:	1. Ballot calculations are performed and the candidate with more than 50% of votes is declared the winner. 2. Winner is displayed on the screen. 3. Audit file is generated.
Main Course:	1. Calculate the number and total percentage of votes received by each candidate 2. Detect which candidate acquired more than 50% of the votes 3. Declare candidate as winner and display their name on the screen 4. Audit File is produced
Alternate Courses:	A possible situation could be that a tie is reached which has to be broken. This has been recorded as a different use case entirely (Use case #9).
Exceptions:	Program Crash 1. In the situation that a program suddenly crashes during vote calculations, the user will have to run the program again. Audit File Write Error: 1. In case of a file write error, the user would have to run the program again from scratch.

Use Case #6: Popularity wins when using IR Voting Protocol

Name:	Popularity wins when using IR Voting Protocol
ID:	6
Description:	The IR voting protocol must determine a winner if a clear majority is not achieved in the first round of ballot calculations.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	1. Ballot calculations are automated and manual labor is reduced. 2. System is more resilient in situations where a clear majority is not established within the first round of ballot calculations
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	Ballots and Candidate information is successfully parsed from

	the CSV file.
Pre-conditions:	1. CSV file is present in the same directory as project 2. CLI is implemented to allow user to enter file name 3. File format is always the same: a. 1st line of file is the voting protocol, which in this case is “IR”. b. 2nd line of the file contains the number of candidates. c. 3rd line of the file contains the candidates and their party separated by commas. d. 4th line contains the number of ballots in the file. e. The rest of the file contains information regarding which candidates were chosen by the voters. 4. There are no errors in the ballots. 5. Each ballot will have at least 1 ranking. 6. The ranking numbers will not have any issues. 7. No write-in candidates. 8. A voter can rank from 1 to the number of candidates. 9. A clear majority has not been reached during the first round of ballot calculations.
Post-conditions:	1. Ballot calculations are performed to determine a winner based on popularity. 2. Winner is displayed on the screen. 3. Audit file is generated.
Main Course:	1. Calculate the number and total percentage of votes received by each candidate. 2. If a clear majority is not determined in the first-round of calculations, the candidate with the lowest number of ballots is eliminated and their votes are redistributed among their second-choice candidates. 3. Steps 1-2 are repeated till a clear majority vote is not established. 4. Declare majority candidate as winner and display their name on the screen. 5. Audit File is produced.
Alternate Courses:	A possible situation could be that a tie is reached which has to be broken. This has been recorded as a different use case entirely (Use case #8).
Exceptions:	Program Crash 1. In the situation that a program suddenly crashes during vote calculations, the user will have to run the program again. Audit File Write Error: 1. In case of a file write error, the user would have to run the

	program again from scratch.

Use Case #7: Implement Open Party List Voting Protocol

Name:	Implement Open Party List Voting Protocol
ID:	7
Description:	This use-case defines how we can use the OPL voting protocol to determine the winner of an election.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Ballot calculations are automated and manual labor is reduced.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	Ballots and Candidate information is successfully parsed from the CSV file.
Pre-conditions:	1. CSV file is present in the same directory as project 2. CLI is implemented to allow user to enter file name 3. File format is always the same: a. 1st line of file is the voting protocol, which in this case is “IR”. b. 2nd line of the file contains the number of candidates. c. 3rd line of the file contains the candidates and their party separated by commas. d. 4th line contains the number of seats. e. 5th line contains the number of ballots. f. File header, Ballots, Number of Seats and Candidate information are successfully parsed. 4. All independents are grouped into 1 party. 5. There are no errors in the ballots. 6. Each ballot will only have one candidate indicated as choice for the vote. 7. No write-in candidates.
Post-conditions:	1. Ballot calculations are performed using OPL voting protocol. 2. Winner is determined and their name is displayed on the screen. 3. Audit file is produced.

Main Course:	1. Determine the total number of votes using the ballot information parsed when reading the CSV file initially. 2. Determine “quota” by dividing the total number of votes by the number of seats to be filled. 3. Divide the number of votes each party received by “quota” to determine the number of seats they can receive in the first round of seat allocation. 4. The parties with the largest remaining number of votes are allocated the remaining seats 5. We repeat step 7 until all parties are assigned at least 1 seat and there are no allocations left to be done. 6. Winner is determined by checking which party has the highest percentage of seats. 7. Winner information is displayed on screen. 8. Audit file is produced.
Alternate Courses:	A possible situation could be that a tie is reached which has to be broken. This has been recorded as a different use case entirely (Use case #8).
Exceptions:	Program Crash 1. In the situation that a program suddenly crashes during vote calculations, the user will have to run the program again. Audit File Write Error: 1. In case of a file write error, the user would have to run the program again from scratch.

Use Case #8: Break a Tie

Name:	Break a Tie
ID:	8
Description:	Regardless of what voting protocol is used, a fair coin toss must be done to determine a winner in the event of a tie.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Tie events are appropriately handled by our application to determine a clear winner.
Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	A Tie occurs once the voting calculations are performed

Pre-conditions:	1. CSV file is processed successfully 2. Ballot calculations according to either of two voting protocols (IR or OPL) have been carried out and resulted in a tie
Post-conditions:	1. Tie is broken and a winner is determined. 2. Winner information is displayed on the screen. 3. Audit File is produced.
Main Course:	1. A tie is determined once ballot calculations are performed. 2. A fair coin toss is done to determine a winner. 3. Winner information is displayed on the screen. 4. Audit File is generated which also contains information about how a tie was achieved and subsequently broken using a fair coin toss.
Alternate Courses:	We have covered all possible courses of action for when a tie occurs so no other alternate course of action is needed.
Exceptions:	Program Crash 1. In the situation that a program suddenly crashes during vote calculations, the user will have to run the program again. Audit File Write Error: 1. In case of a file write error, the user would have to run the program again from scratch.

Use Case #9: Display winner information

Name:	Display winner and election information
ID:	9
Description:	Once all the ballot calculations are completed and a winner is determined based on our selected voting protocol, we can display the winner information on screen along with election information such as the type of election, number of seats, number of ballots cast as well as a statistical breakdown of how many votes were received by each candidate.
Actors:	Election Officials, Programmers and Testers.
Organizational Benefits:	Users of the application will have a way to test if their system correctly counts the votes and determines the winner given different voting protocols or tie-break scenarios.

Frequency of Use:	Multiple times during the year at normal election times and special elections.
Triggers:	The election has concluded with a clear winner being determined after ballot calculations are completed.
Pre-conditions:	1. CSV file is successfully input by the user 2. CSV file is successfully parsed by the program 3. Ballot calculations are performed and a winner is determined.
Post-conditions:	1. Winner and election information is displayed on screen to the user.
Main Course:	1. Ballot calculations are complete and a winner is determined 2. Information about the winning candidate is received such as % of ballots and/or seats received and candidate/party information. 3. Information about the type of election, number of seats, and number of ballots cast for each candidate should also be displayed on screen.
Alternate Courses:	Any system errors should cause the program to exit gracefully.
Exceptions:	Program Crash 1. In the situation that a program suddenly crashes while displaying winner information, the user will have to run the program again.