Software Design Document

Voting System

3 Required Diagrams are in Section 8.1 (see Table of Contents)

Prepared by Team3:
Shi Chen (chen4264)
Dong Liang (liang492)
Song Liu (liux4169)
Qing Hong (hong0465)

10/28/2018

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The purpose of this software design document (SDD) is to describe the system architecture and design of the Voting System. The Voting System is designed to read a CSV file provided by the user containing ballots and execute either an Open Party Listing (OPL) election or an Instant Runoff election (IR).

The intended audience is the developers of the Voting System, under the umbrella of Team3. This includes all software developers and testers on Team3. This will serve as a guide in the development stage and is designed to satisfy all requirements specified in the SRS for the Voting System.

1.2 Scope

The software is a standalone system that will be used by election officials. It will run on either Linux or Windows systems. The goal is to provide an efficient and automated of determining election results with two different methods. One is to perform an Instant Runoff election and provide the results and an audit of the election. The other is to perform an Open Party Listing election and provide the results and an audit for that type of election. The benefits of this software are to limit the manual labor to determine election results and to maintain transparency in the election process.

1.3 Overview

In the Appendix section 8.1 Required Diagrams, is where all the required diagrams with descriptions are placed (also embedded as figures in the document).

Section 1 provides a summary of this document's purpose and the scope of the Voting System. Section 2 provides a brief description of the system context and the functionalities of the Voting System. Section 3 provides a decomposition of the Voting System into its class components and the operation of the system. Section 4 and 5 describes the entities and their attributes and functions. Section 6 provides the user interface as well as the design of the output files.

Note that we placed the low level details of the classes in Sections 4 and 5 while Section 3 is primarily for the diagram descriptions. This was done to consolidate information in single spots since Section 4 and 5 requires descriptions of the attributes and functions.

1.4 Reference Material

I.E.E.E Software Design Document Template:

- https://ay17.moodle.umn.edu/pluginfile.php/2836810/mod_resource/content/1/sdd_template.pdf

Detailed Explanation of Instant Runoff Voting at FairVote

- https://www.fairvote.org/plurality_majority_systems

Detailed Explanation of Open Party Listing Voting at FairVote

- https://www.fairvote.org/proportional_representation_voting_systems

SRS for Voting System

- https://github.umn.edu/umn-csci-5801-F18/repo-Team3/blob/master/SRS/SRS_Team3.pdf

1.5 Definitions and Acronyms

Term	**Definition**
SRS	Software Requirements Specification. Details the functional and nonfunctional requirements of the system. The SRS document for the Voting System is provided in the references (Section 1.4).
SDD	Software Design Document
OPL	Open Party Listing. A method of voting. See the references in Section 1.4 for more information.
IR	Instant Runoff. A method of voting. See the references in Section 1.4 for more information.
CSV	Comma Separated Values. Is a file type. It separated values using commas. It is the file type of the files Voting System will be reading and will provide the ballots of the election.
OOP	Object Oriented Programming.

2. SYSTEM OVERVIEW

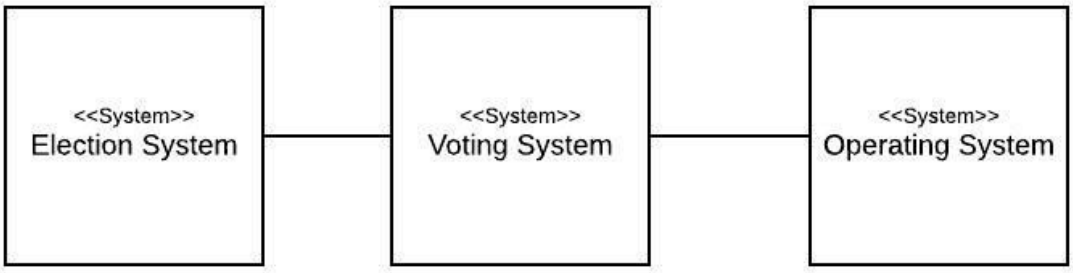


Figure 1: The context model of the Voting System

Figure 1 displays the context of the Voting System. The Election System is an external system that will handle the actual election process (i.e. collecting the ballots and votes) itself and will provide the input CSV files. The Voting System (the system detailed in this SDD) is responsible for reading the provided CSV files and performing the IR and OPL voting methods and displaying the results of the elections and providing summary files and audit files for both types of elections. The files will be written to the Operating System where users may extract them.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

Currently, the system is divided into several components.

The **Candidate** class tracks the information for a candidate in an election.

LogUtil is an interface which is implemented differently for each voting differently.

OPLVoting is a realization of the LogUtil interface. It will perform the logic of the Open Party Listing voting method.

IRVoting is a realization of the LogUtil interface. It will perform the logic of the Instant Runoff voting.

VoteCounter is a class that will handle input of file name.

Util will handle reading the CSV file.

The **Party** class will store party information for an OPL election.

Ballot class will store the information of the ballots.

Figure 2 (below) shows the relations between different classes and interface. VoteCounter, IRVoting and OPLVoting will implement the LogUtil interface so they can report logs properly. In addition, Candidate class has direct association with Ballot class and party class. IRVoting class has association with Candidate and Ballot, and OPLVoting class has association with Ballot class, Candidate class and Party class. The system will initiate either an IRVoting instance or an OPLVoting instance to solve the given problem according to the request info in the given file.

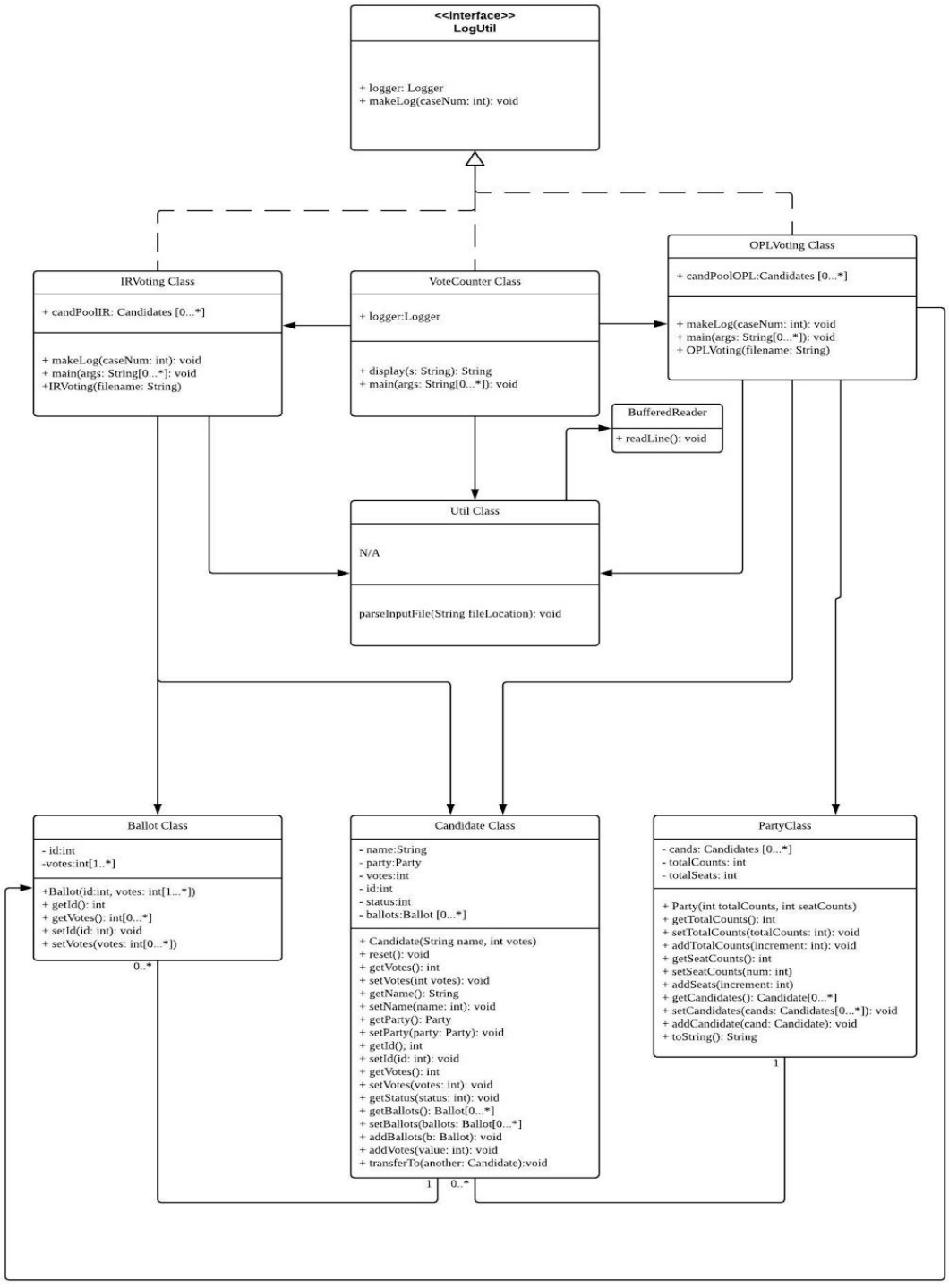


Figure 2: Class Diagram of the System

3.2 Decomposition Description

This section will decompose the system further and highlight how the system runs.

3.2.1 Activity Diagram for Running the System

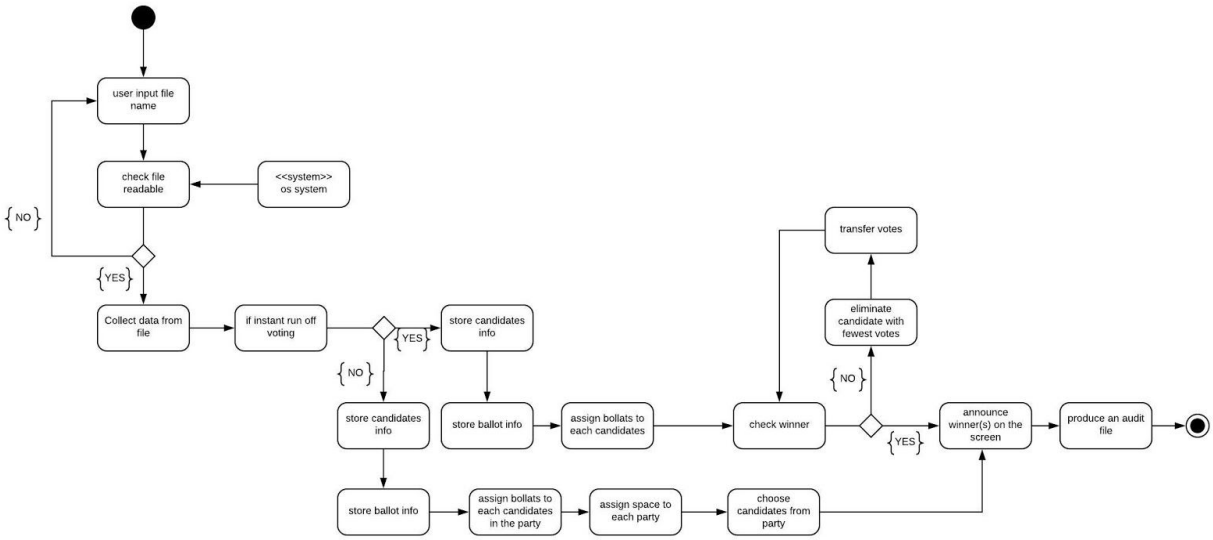


Figure 3: Activity Diagram for OPL and IR

Figure 3 shows the system will ask for input file until the file is valid. And it collects data from the file. If the voting information is not instant runoff system which means it is open party list system. When all the data was stored in the database, it will calculate the proportion of votes to each party and assign the total space to each party. In the end, the candidates from each party are selected.

Otherwise it is instant runoff system, the system will store all the data information in the file to the database. And after that it will check the first round winner, if there is no winner, it will eliminate the candidate win fewest votes and transfer the votes until the winner appear. When the winner appears, it will display on the screen along with the information. Then an audit file will be created with the election information at the time.

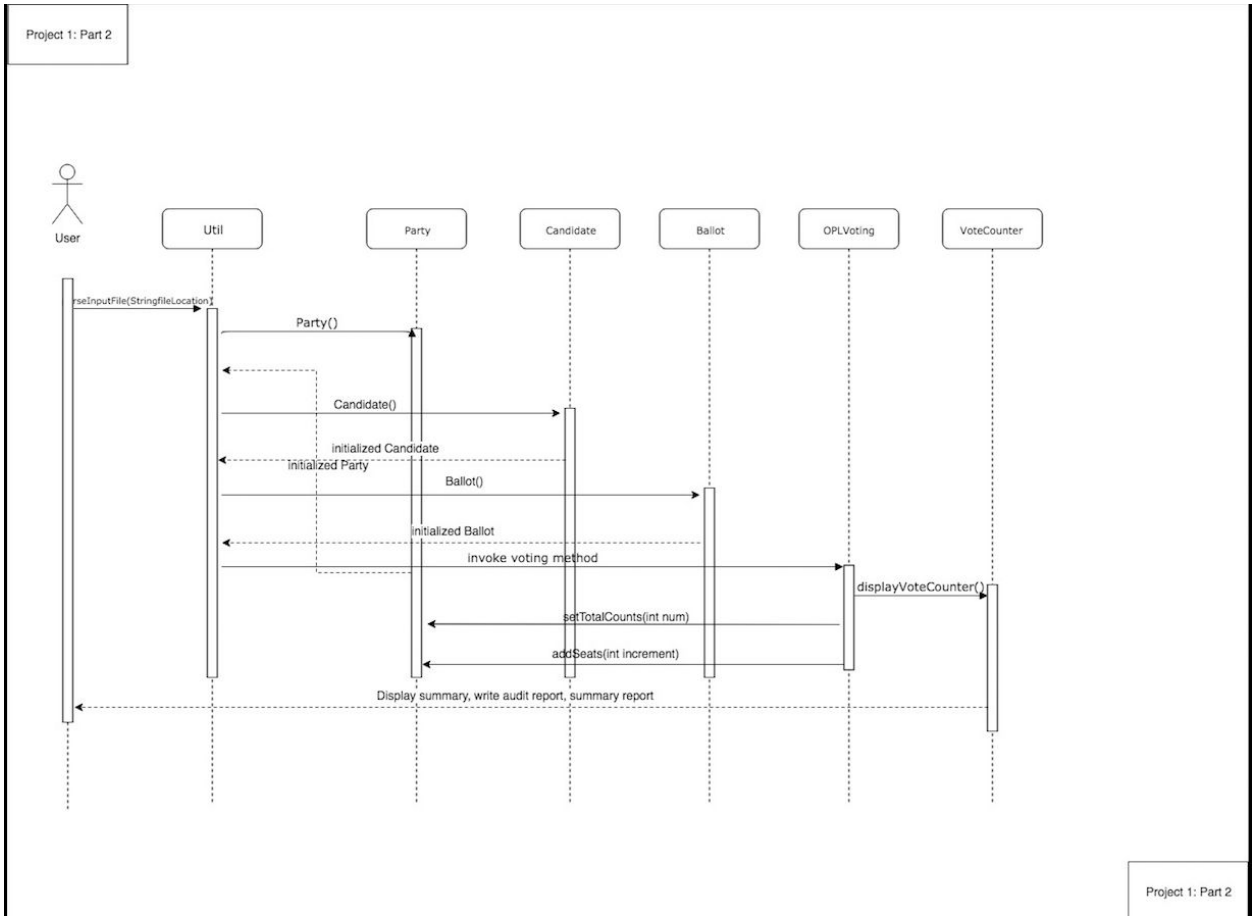


Figure 4: Sequence Diagram for OPL

3.2.2 Sequence Diagram for OPL Voting

Figure 4 shows the process of Open Party List Voting to allocate seats. User inputs voting file to the system, then the Util class would parse the file and get related Party, Candidate and Ballot data. After the file is parsed, the system would invoke methods of OPLVoting class which would implement OPL voting process to get election result. When the election result is determined after processing all voting data, VoteCounter would display the result of the election.

3.2.3 Use Cases

This section contains the use cases for the System.

UC_001

Name	Official passes name of an Instant Runoff file to program.
ID	UC_001

Description	We prompt the user for the file name. A terminal input of the file name is typed in by the user.
Actors	Election Officials
Organizational Benefits	The election officials are able to determine the winner of an election without manual calculations.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	Program is built and active.
Preconditions	1. Election file is in the root directory as the program. 2. Election file is marked “IR” on 1st line. 3. File is a csv format.
Post Conditions	1. The file is read into the program and ready for the Instant Runoff method.
Main Course	1. Program asks user for file name using string output on the terminal. 2. User types in file name and hit enter.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_002

Name	Official passes name of an Open Party Listing file to program.
ID	UC_002
Description	We prompt the user for the file name. A terminal input of the file name is typed in by the user.
Actors	Election Officials
Organizational Benefits	The election officials are able to determine the winner of an election without manual calculations.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	Program is built and active.
Preconditions	1. Election file is in the root directory as the program. 2. Election file is marked “OPL” on 1st line. 3. File is a csv format.

Post Conditions	The file is read into the program and ready for the Open Party Listing algorithm.
Main Course	1. Program asks user for file name using string output on the terminal. 2. User types in file name and hits enter.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_003

Name	Election Officials run the Instant Runoff voting method.
ID	UC_003
Description	Election Official passes in a file of an Instant Runoff election and the election results are generated.
Actors	Election Officials
Organizational Benefits	Election results can be calculated without manual effort.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	File is in the same directory as the program. File has “IR” written on the first line.
Post Conditions	1. An audit file of the Instant Runoff is available to the user in the same directory as the input file. 2. A summary file of the Instant Runoff available to the user in the same directory as the input file.
Main Course	1. Instant Runoff election is performed. 2. An audit file of the Instant Runoff election is written to the directory of the program and visible to official in the directory. 3. Official views summary results on terminal screen 4. Summary report is written to directory of the program and visible to the official in the directory.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_004

Name	Election Officials run the Open Party Listing voting method.
ID	UC_004
Description	Election Official passes in a file of an Open Party Listing election and the election results are generated.
Actors	Election Officials
Organizational Benefits	Election results can be calculated without manual effort.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_002).
Preconditions	File is in the same directory as the program. File has “OPL” written on first line
Post Conditions	1. An audit file of the Open Party Listing election is available to the user in the same directory as the input file. 2. A summary file of the Open Party Listing is available to the user in the same directory as the input file (see UC_005).
Main Course	1. Open Party Listing election is performed. 2. An audit file of the Open Party Listing election is written to the directory of the program and visible to official in the directory. 3. Official views summary results on terminal screen 4. Summary report is written to directory of the program and visible to the official in the directory.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_005

Name	Generate Instant Runoff election summary reports.
ID	UC_005
Description	When an Election Official runs an Instant Runoff election, a summary report is generated. It contains summary results like the type of the election, the winners, and number seats, etc.

Actors	1. Election Official Related: Media personnel will receive the election results.
Organizational Benefits	The results of an Instant Runoff election are summarized succinctly and clearly.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has “IR” on line 1.
Post Conditions	A summary report of the Instant Runoff election is in the same directory as the election file.
Main Course	1. A summary report is written to the same directory as the election file and visible to the election official and accessible to the official. a. File should show the: i. Type of election ii. Number of ballots cast iii. Number of votes received for each candidate. iv. The winner of the election.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_006

Name	Generate Open Party Listing election summary reports.
ID	UC_006
Description	When an Election Official runs an Open Party Listing election, a summary report is generated. It contains summary results like the type of the election, the winners, and number seats, etc.
Actors	1. Election Official Related: Media personnel will receive the election results.
Organizational Benefits	The results of an Open Party Listing election are summarized succinctly and clearly.
Frequency of Use	Multiple times during an election season or special elections.

Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has “OPL” on line 1.
Post Conditions	A summary report of the Open Party Listing election is in the same directory as the election file.
Main Course	1. A summary report is written to the same directory as the election file and accessible to the user. a. File should show the: i. Type of election ii. Number of seats in the election iii. Number of ballots cast iv. Number of votes received for each candidate v. Winners of the election vi. Seats won by each party
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_007

Name	Display a summary of an Instant Runoff election to the Election Official on the screen.
ID	UC_007
Description	When an Election Official runs an Instant Runoff election, a summary of the election is displayed to the user with information like the winners, the type of the election, etc.
Actors	1. Election Official
Organizational Benefits	The results of an Instant Runoff election are summarized and displayed succinctly and clearly.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has “IR” on line 1.
Post Conditions	A summary report of the Instant Runoff election is in the same directory as the election file.

Main Course	1. A summary report is displayed to the election official on the screen. a. It should show: i. Type of election ii. Number of ballots cast iii. Number of votes received for each candidate. iv. Winner of the election
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_008

Name	Display a summary of an Open Party Listing election to the Election Official on the screen.
ID	UC_008
Description	When an Election Official runs an Instant Runoff election, a summary of the election is displayed to the user with information like the winners, the type of the election, the number of seats won by each party, etc.
Actors	Election Official
Organizational Benefits	The results of an Open Party Listing election are summarized and displayed succinctly and clearly.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has “OPL” on line 1.
Post Conditions	A summary report of the Open Party Listing is displayed on the screen.
Main Course	1. A summary report is displayed to the election official on the screen. a. It should show: i. Type of election ii. Number of ballots cast iii. Number of votes received for each candidate.

	iv. Winners of the election v. Number of seats in the election
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_009

Name	Election official receives an audit report for an Instant Runoff election.
ID	UC_009
Description	When an Election Official runs an Instant Runoff election, an audit report is generated with details such as the winner and how the election progressed.
Actors	Election Official
Organizational Benefits	Helps ensure transparency of the election process.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has “IR” on line 1.
Post Conditions	An audit report of the Instant Runoff election is in the same directory as the election file.
Main Course	1. An audit report is written to the directory of the election file and accessible to the user. a. It should show: i. Instant Runoff Election ii. Number of ballots cast iii. Number of votes received for each candidate. iv. Winner of the election v. Calculations to determine the winner. vi. Progression of the election
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_010

Name	Election official receives an audit report for an Open Party Listing election.
ID	UC_010
Description	When an Election Official runs an Open Party Listing election, an audit report is generated with details such as the winner and how the election progressed.
Actors	Election Official
Organizational Benefits	Helps ensure transparency of the election process.
Frequency of Use	Multiple times during an election season or special elections.
Triggers	User passes in a csv file name to the program (see UC_001).
Preconditions	1. Election file is in the same directory as the program. 2. Election file has "OPL" on line 1.
Post Conditions	An audit report of the Open Party Listing election is in the same directory as the election file.
Main Course	1. An audit report is written to the directory of the election file and accessible to the user. It should show: v. Open Party Listing vi. Number of candidates and their names vii. Number of ballots cast viii. Number of votes received for each candidate. ix. Winners of the election x. Calculations to determine the winners. xi. Progression of the election.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_011

Name	Programmer tests inputting Instant Runoff file name.
ID	UC_011
Description	The programmer should be able to test the inputting of the file

	name for an Instant Runoff election.
Actors	Programmer
Organizational Benefits	The correctness of the system is ensured.
Frequency of Use	Frequently in the Implementation and Verification Stages
Triggers	Program is built and active.
Preconditions	4. Election file is in the root directory as the program. 5. Election file is marked “IR” on 1st line. 6. File is a csv format.
Post Conditions	1. File is read into the program.
Main Course	1. Program asks programmer for file name using string output on the terminal. 2. Programmer types in file name and hit enter. 3. Programmer verfies file has been read properly.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_012

Name	Programmer tests inputting Open Party Listing file name.
ID	UC_012
Description	The programmer should be able to test the inputting of the file name for an Open Party Listing election.
Actors	Programmer
Organizational Benefits	The correctness of the system is ensured.
Frequency of Use	Frequently in the Implementation and Verification Stages
Triggers	Program is built and active.
Preconditions	1. Election file is in the root directory as the program. 2. Election file is marked “OPL” on 1st line. 3. File is a csv format.
Post Conditions	1. File is read into the program.

Main Course	4. Program asks programmer for file name using string output on the terminal. 5. Programmer types in file name and hit enter. 6. Programmer verfies file has been read properly.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_013

Name	Programmer or tester tests Instant Runoff election.
ID	UC_013
Description	The tester should be able to test the correctness of the Instant Runoff election
Actors	Testers and programmers
Organizational Benefits	The correctness of the system is ensured.
Frequency of Use	Frequently in the Implementation and Verification Stages
Triggers	Tester or programmers passes in an Instant Runoff file name for testing.
Preconditions	1. Election file is in the root directory as the program. 2. Election file is marked “IR” on 1st line. 3. File is a csv format.
Post Conditions	1. Audit and summary file are in file directory. 2. Summary results are displayed to the screen.
Main Course	1. Tester/programmer verifies correctness of visual summary. 2. Tester/programmer verifies correctness of audit report. 3. Tester/programmer verifies correctness of summary report.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

UC_014

Name	Programmer or tester tests Open Party Listing election.
ID	UC_014

Description	The tester should be able to test the correctness of the Open Party Listing election
Actors	Testers and programmers
Organizational Benefits	The correctness of the system is ensured.
Frequency of Use	Frequently in the Implementation and Verification Stages
Triggers	Tester or programmers passes in an Open Party Listing file name for testing.
Preconditions	1. Election file is in the root directory as the program. 2. Election file is marked “OPL” on 1st line. 3. File is a csv format.
Post Conditions	1. Audit and summary file are in file directory. 2. Summary results are displayed to the screen.
Main Course	1. Tester/programmer verifies correctness of visual summary. 2. Tester/programmer verifies correctness of audit report. 3. Tester/programmer verifies correctness of summary report.
Alternate Courses	None
Exceptions	This iteration will have no file or input errors.

3.3 Design Rationale

The design rationale stemmed primarily from the programming language requirement of either C++ or Java, both of which lend themselves towards an object oriented design. The distinction of the two voting methods delineated a clear separation in interfaces and showed clear entities such as Candidate, Party, and Ballot.

4. DATA DESIGN

4.1 Data Description

The system is designed in an OOP paradigm. The CSV being read in will be read using the Java i.o class BufferedReader and all balloting and candidate information will be read using BufferedReader. The balloting information will then be stored in the classes described in Section 3 (Ballot for ballots, Candidate for candidates, and Party for parties).

4.2 Data Dictionary

The description of the functions will be in Section 5 to avoid duplication of information.

4.2.1 Ballot

An object class used to hold information on ballots provided in the lines of the CSV input file.

Member attributes	**Description**
id: int	Identification number of the Ballot.
votes: int []	An integer array representing the markings on a Ballot.

4.2.2 Candidate

An object class used to hold information on Candidates provided in the lines of the CSV input file.

Member attributes	**Description**
id: int	Unique identification number of the Candidate.
name: String	String name of Candidate.
party: String	Name of Candidate's Party affiliation (if applicable)
votes: int	Count for number of votes (Ballots) received.
status: int	Status in the election. 0 is eliminated and 1 is alive.

4.2.3 IRVoting

Class responsible for running the IR voting method.

Member attributes	**Description**

candPoolIR: ArrayList<Candidates>	Container for all Candidates involved in the election.

4.2.4 LogUtil

Interface for logging messages and is inherited by the OPLVoting and IRVoting.

Member attributes	**Description**
logger: Logger	A Java object for logging messages.

4.2.5 OPLVoting

Class responsible for running the OPL voting method.

Member attributes	**Description**
candPoolOPL: ArrayList<Candidates>	Container for all Candidates involved in the election.

4.2.6 Party

Class responsible for holding Party information the OPL voting method.

Member attributes	**Description**
candidates: String[]	Names of Candidates representing the Party.
seatCounts: int	Number of seats won by the Party
totalCounts: int	Total Ballots won by the Party.

4.2.7 Util

Class responsible for file handling.

Member attributes	**Description**
None	None

4.2.8 VoteCounter

Main driver of the program.

Member attributes	Description
logger: Logger	A Java object for logging messages.

5. COMPONENT DESIGN

5.1 VoteCounter

Name	main()
Input	None
Output	None
Description	The driver of the program. Will prompt the user for a user input filename.

5.2 Util

Name	parseInputFile(String filename)
Input	filename: The String name of the file.
Output	None
Description	Parses the CSV file referenced by the filename passed in by the user.

5.3 LogUtil

An interface for logging messages.

Name	makeLog(int caseNum)
Input	caseNum: the case number of the message
Output	None
Description	Creates a log for this specific caseNum.

5.4 IRVoting

Name	makeLog(int caseNum)
Input	caseNum: the case number of the message
Output	None
Description	Creates a log for this specific caseNum.

Name	IRVoting(String filename)
Input	filename: The String name of the file.
Output	None
Description	Handles the logic of the IR election method. That includes the distribution of transfer of votes during runoff elections, the breaking of ties, and writing to the summary files and audit files and displaying to the screen.

5.4.1 IRVoting Pseudocode

```
IRVoting::IRVoting(String file_name)
        let audit_file be a new text file
        let summary_file be a new text file
        let reader be a reader for file_name
        line_number <- 1
        line <- reader.readLine()
        num_candidates <- 0
        num_ballots <- 0
        winner_found <- FALSE
        while line != NIL
                if line_number == 1
                        if line == OPL
                                return
                        else
                                write <“ Election Type: IR”> to audit_file
                                write < “Election Type: IR”> to summary_file
                else if line_number == 2
                        num_candidates <- Integer.parseInt(line)
                        write <”Number of Candidates: ” num_candidates>, to audit_file
                        write <”Number of Candidates: ” num_candidates>, to summary_file
                else if line_number == 3
                        candidates <- line.Split(“,”)
                        for candidate: candidates
                                create a Candidate instance
```

```
                                        write <candidate name> to audit_file and summary_file
            else if line_number == 4
                        num_ballots <-  Integer.parseInt(line)
                        write <”Number of Ballots: ” num_ballots>, to audit_file
                        write <”Number of Ballots: ” num_ballots>, to summary_file
            else
                        let B be Ballot for line
                        let C be Candidate ranked #1on B
                        C.addBallot(B)
                        write <Candidate C gets ballot B> to audit_file
end
let C be Candidate with most votes
if C.getVotes() > num_ballots/2 OR num_candidates == 2
            winner_found <- TRUE
            C is the winner of the election
            write <C is the winner> to audit_file, summary_file
for each Candidate
            write <number of votes each candidate received> to audit_file and summary_file
else
            while winner_found == FALSE && num_candidates > 2
                        let C be Candidate with the least amount of votes
                        C.setStatus(0)
                        num_candidates -= 1
                        write <C is eliminated> to audit file
                        for each Candidate C’ still in election
                                    C.transferTo(C’)
                                    <write ballot is transferred from C to C’>
                        let C’’ be Candidate with most votes
                        if C’’.getVotes() > num_ballots/2
                                    winner_found <- TRUE
                                    C’’ is the winner of the election
                                    write <C’’ is the winner> to audit_file, summary_file
                        for each Candidate
                                    write <number of votes each candidate received> to audit_file and summary_file
if winner_found == FALSE and num_candidates == 2
            let A, B be remaining Candidates
            if A.getVotes()>B.getVotes()
                        A is the winner of the election.
                        write <A is the winner> to audit_file, summary_file
            if A.getVotes()<B.getVotes()
                        B is the winner of the election.
                        write <B is the winner> to audit_file, summary_file
            else
                        let C be the winner chosen randomly from A and B
                        C is the winner of the election.
                        write <C is the winner> to audit_file, summary_file
```

5.5 OPLVoting

Name	makeLog(int caseNum)
Input	caseNum: the case number of the message

Output	None
Description	Creates a log for this specific caseNum.

Name	OPLVoting(String filename)
Input	filename: The String name of the file.
Output	None
Description	Handles the logic of the OPL election method. That includes the calculation of the quota, the distribution of seats to the parties, the distribution of leftover seats, the breaking of ties, and writing to the summary files and audit files and displaying to the screen.

5.5.1 OPLVoting Pseudocode

OPL Summary Report
Election Type: OPL
Number of Candidates: <number of candidates>
Number of Seats: <number of seats>
Number of Ballots: <number of ballots>
Party <party name> won <number of seats> seats
Candidate <Candidate Name> won seat.
.
.
Party <party name> won <number of seats> seats
Candidate <Candidate Name> won seat
.
.
Candidate <Candidate Name> received <number of votes>
Candidate <Candidate Name> received <number of votes>
.
.

5.6 Candidate

Name	transferTo(Candidate another)
Input	another: Candidate this candidate is going to transfer his ballots to.

Output	None
Description	This current instance is being eliminated in the IR election and his Ballots will be transferred to Candidate another.

Name	getVotes()
Input	None
Output	*votes*: The number of votes/ballots received.
Description	Returns the vote count (number of ballots won) for the Candidate.

Name	setVotes(int value)
Input	value: The new vote count.
Output	None
Description	Set *votes* parameter to new value.

Name	addVotes(int value)
Input	value: Increment vote count by this amount.
Output	None
Description	Increment *votes* parameter to new value.

Name	getBallots()
Input	None
Output	*ballots*: Container with all Ballots won by this Candidate.

Description	Returns the Ballots won by this Candidate.

Name	setBallots(Ballot [] ballots)
Input	ballots: The new set of Ballots.
Output	None
Description	Set *ballots* parameter to a new value.

Name	addVotes(int value)
Input	value: Increment vote count by this amount.
Output	None
Description	Increment *votes* parameter to new value.

Name	toString()
Input	None
Output	String value displaying Candidate name, number of votes, and election status.
Description	Prints String of Candidate name, number of votes, and election status.

Name	getParty()
Input	None
Output	*party*: The Party the Candidate belongs to.
Description	Returns the String Party name the Candidate belongs to.

Name	setParty(String party)
Input	party: The new Party to set Candidate's *party* to.
Output	None
Description	Set *party* parameter to a new value.

Name	getId()
Input	None
Output	*id*: The identifier of the Candidate.
Description	Returns the *id* parameter of the Candidate.

Name	setId(Party id)
Input	id: The new id to set Candidate's *id.*
Output	None
Description	Set *id* parameter to a new value.

Name	getStatus()
Input	None
Output	*status*: The status of the Candidate: 0 is eliminated and 1 is still in the election.
Description	Returns the *id* parameter of the Candidate.

Name	setStatus(int status)
Input	status: The new status to set Candidate's *status.*

Output	None
Description	Set *status* parameter to a new value (0 or 1).

Name	getName()
Input	None
Output	*name*: The String name of the Candidate.
Description	Returns the *name* parameter of the Candidate.

Name	setName(String name)
Input	name: The new name to set Candidate's *name.*
Output	None
Description	Set *name* parameter to a new value.

5.7 Party

Name	setTotalCounts(int num)
Input	num: The new vote count to set the Party vote total to.
Output	None
Description	Set *totalCounts* parameter to a new value.

Name	setTotalCounts(int num)
Input	num: The new vote count to set the Party vote total to.

Output	None
Description	Set *totalCounts* parameter to a new value.

Name	getTotalCounts()
Input	None
Output	*totalCounts*: The total vote count for all Candidates of the Party.
Description	Returns the *totalCounts* parameter of the Party.

Name	setSeatCounts(int num)
Input	num: The new vote count to set the Party *seatCounts* parameter to.
Output	None
Description	Set *seatCounts* parameter to a new value.

Name	getSeatCounts()
Input	None
Output	*seatCounts*: The seats received by the Party.
Description	Returns the *seatCounts* parameter of the Party.

Name	addSeats(int increment)
Input	increment: Increment *seatCounts* by this amount.

Output	None
Description	Increment *seatCounts.*

Name	addTotalCounts(int increment)
Input	increment: Increment *totalCounts* by this amount.
Output	None
Description	Increment *totalCounts.*

Name	getCandidates()
Input	None
Output	*cands*: Container with all Candidates of the Party.
Description	Returns the Candidates of the Party

Name	setCandidates(Candidate [] cands)
Input	cands: The new set of Candidate names.
Output	None
Description	Set *cands* parameter to a new value.

Name	toString()
Input	None
Output	String information of the Party including Candidate names.
Description	Returns String information of the Party.

5.8 Ballots

Name	getVotes()
Input	None
Output	*votes*: The array encompassing the ballot values.
Description	Return *votes*, the balloting information.

Name	setVotes(int [] votes)
Input	votes: The array encompassing new ballot values.
Output	None
Description	Reset *votes*, the balloting information.

Name	getId()
Input	None
Output	*id*: The identification number of the Ballot.
Description	Return *votes*, the balloting information.

Name	setId(int id)
Input	id: The new identification number
Output	None
Description	Reset *id*, the identification information.

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user will be operating the software from the command prompt. The system will prompt the user to provide a file name at which point the user will type in the file name and hit enter. Future versions may provide a graphical user interface.

6.2 Screen Images

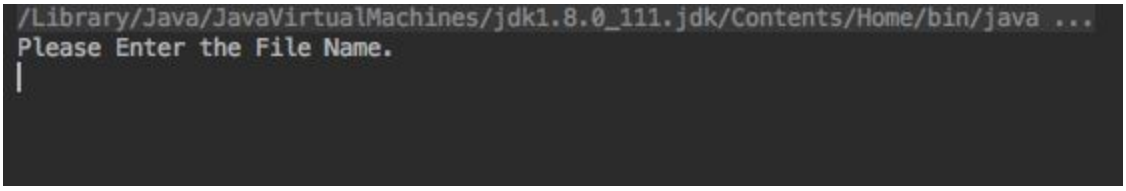


Figure 5: A screenshot of the terminal.

Figure 5 shows the terminal output that the user will see at the beginning of the program. It is simply a text prompt asking for a file name which the user will enter.

6.3 Screen Objects and Actions

Currently, the system is terminal based and there are no screen objects.

6.4 File Templates

This is how the IR summary and screen report will look.

IR Summary Report
Election Type: IR
Number of Candidates: <number of candidates>
Number of Ballots: <number of ballots>
Candidate <Winner> is the winner.
Candidate 1: <Candidate Name> received <number of votes>
Candidate 2: <Candidate Name> received <number of votes>
.
.
.

This is how the OPL summary and screen report will look.

OPL Summary Report
Election Type: OPL
Number of Candidates: <number of candidates>
Number of Seats: <number of seats>
Number of Ballots: <number of ballots>
Party <party name> won <number of seats> seats
Candidate <Candidate Name> won seat.
.
.
Party <party name> won <number of seats> seats
Candidate <Candidate Name> won seat
.

.
Candidate <Candidate Name> received <number of votes>
Candidate <Candidate Name> received <number of votes>
.
.

.
.

This is how the IR audit file will look like.

IR Audit File Template

Election Type: IR
Number of Candidates: <number of candidates>
Candidate 1: <Candidate Name>
.
.
candidates
.
.
Number of Ballots: <num_ballots>
.
.
Ballot <number> is won by Candidate <Candidate Name>
.
.
Candidate C won election with <number of votes> votes (if majority is won already)
Candidate <Candidate Name> won <number of votes> votes
Candidate <Candidate Name> won <number of votes> votes
.
.
candidates
.
.
(If no majority)
Runoff Round 1
Candidate C was eliminated.
Ballot B1 was reassigned to Candidate C'
Ballot B2 was reassigned to Candidate C''
.
.
Candidate <Candidate Name> won <number of votes> votes
Candidate <Candidate Name> won <number of votes> votes
.
.
(after however many rounds)
.
.
Candidate C won election with <number of votes> votes (if majority is won already)
Candidate <Candidate Name> won <number of votes> votes
Candidate <Candidate Name> won <number of votes> votes

This is how the OPL audit file will look like.

OPL Audit File Template
Election Type: OPL
Number of Candidates: <number of candidates>
Party 1: <Party Name>
.
.
parties
.
.
Candidate 1: <Candidate Name>
.
.
candidates
.
.
Number of Ballots: <number of ballots>
.
.
Number of Seats: <number of seats>
.
.
Candidate <C> of Party <P> gets Ballot <B>
Candidate <C> of Party <P> gets Ballot <B>
.
.
Quota was <number of ballots> / <number of seats>
Party P won <n> ballots
Party <P> received <n/quota> seats
Party <P> has <n%quota> leftover ballots
.
.
Party <P'> won <n'> ballots
Party <P'> received <n'/quota> seats
Party <P'> has <n'%quota> leftover ballots
.
.
Party <P''> received <n''> leftover seats
Party <P'''> received <n'''> leftover seats
.
.
Party <P1> received <n1> votes
Candidate <Candidate Name> of <P1> won seat.
Candidate <Candidate Name> of <P1> won seat.
.
.
Candidate <Candidate Name> of <P1> received <c1> seats
Candidate <Candidate Name> of <P1> received <c2> seats
.
.
Party <P2> received <n2> votes
Candidate <Candidate Name> of <P2> won seat.
Candidate <Candidate Name> of <P2> won seat.
.
.

Candidate <Candidate Name> of <P2> received <d1> seats *Candidate <Candidate Name> of <P2> received <d2> seats*

7. REQUIREMENTS MATRIX

Requirement	Description	Sections
4.1.3:REQ-1	User needs to pass in an Instant Runoff election filename.	Figure 3 Section 5.2
4.1.3:REQ-2	User needs to pass in Open Party Listing election filename.	Figure 3 Section 5.2
4.2.3:REQ-1	User performs an Instant Runoff election.	Figure 3 Section 5.2 Section 5.4 and 5.4.1
4.2.3:REQ-1.1	System breaks tie in Instant Runoff randomly.	Section 5.4 and 5.4.1
4.2.3:REQ-2	System generates an audit report of Instant Runoff election for the user.	Figure 3 Section 5.4 and 5.4.1 Section 6.4
4.2.3:REQ-2.1	The IR audit report must provide basic information for the user.	Section 5.4 and 5.4.1 Section 6.4
4.2.3:REQ-2.2	The IR audit report must show the entire process of an Instant Runoff election.	Section 5.4 and 5.4.1 Section 6.4

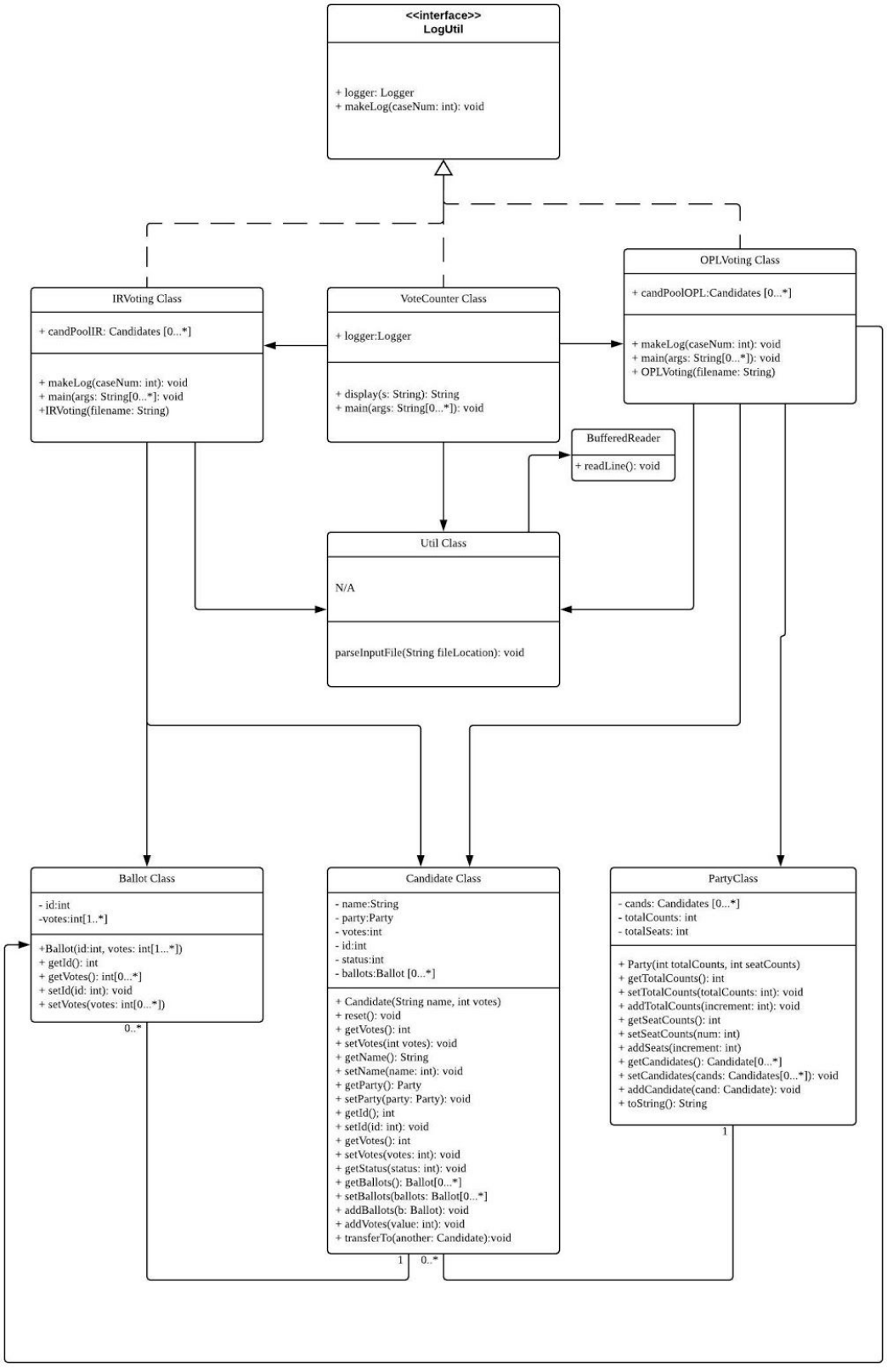
4.2.3:REQ-3	System generates a summary report of an Instant Runoff election for the user.	Section 5.4 and 5.4.1 Section 6.4
4.2.3:REQ-4	System displays results of Instant Runoff election to the user.	Section 5.4 and 5.4.1 Section 6.4
4.3.3:REQ-1	User performs an Open Party Listing election.	Figure 3 Figure 4 Section 5.5 and 5.5.1
4.3.3:REQ-1.1	System breaks tie in Open Party Listing randomly.	Section 5.5 and 5.5.1
4.3.3:REQ-2	System generates an audit report of Open Party Listing election for the user.	Figure 3 Figure 4 Section 5.5 and 5.5.1 Section 6.4
4.3.3:REQ-2.1	The OPL audit report must provide basic information for the user.	Section 5.5 and 5.5.1 Section 6.4
4.3.3:REQ-2.2	The OPL audit report must show the entire process of an OPL election.	Section 5.5 and 5.5.1 Section 6.4

4.3.3:REQ-3	System generates a summary report of an OPL election for the user.	Section 5.5 and 5.5.1 Section 6.4
4.3.3:REQ-4	System displays results of OPL election to the user.	Section 5.5 and 5.5.1 Section 6.4

8. APPENDICES

8.1 Required Diagrams

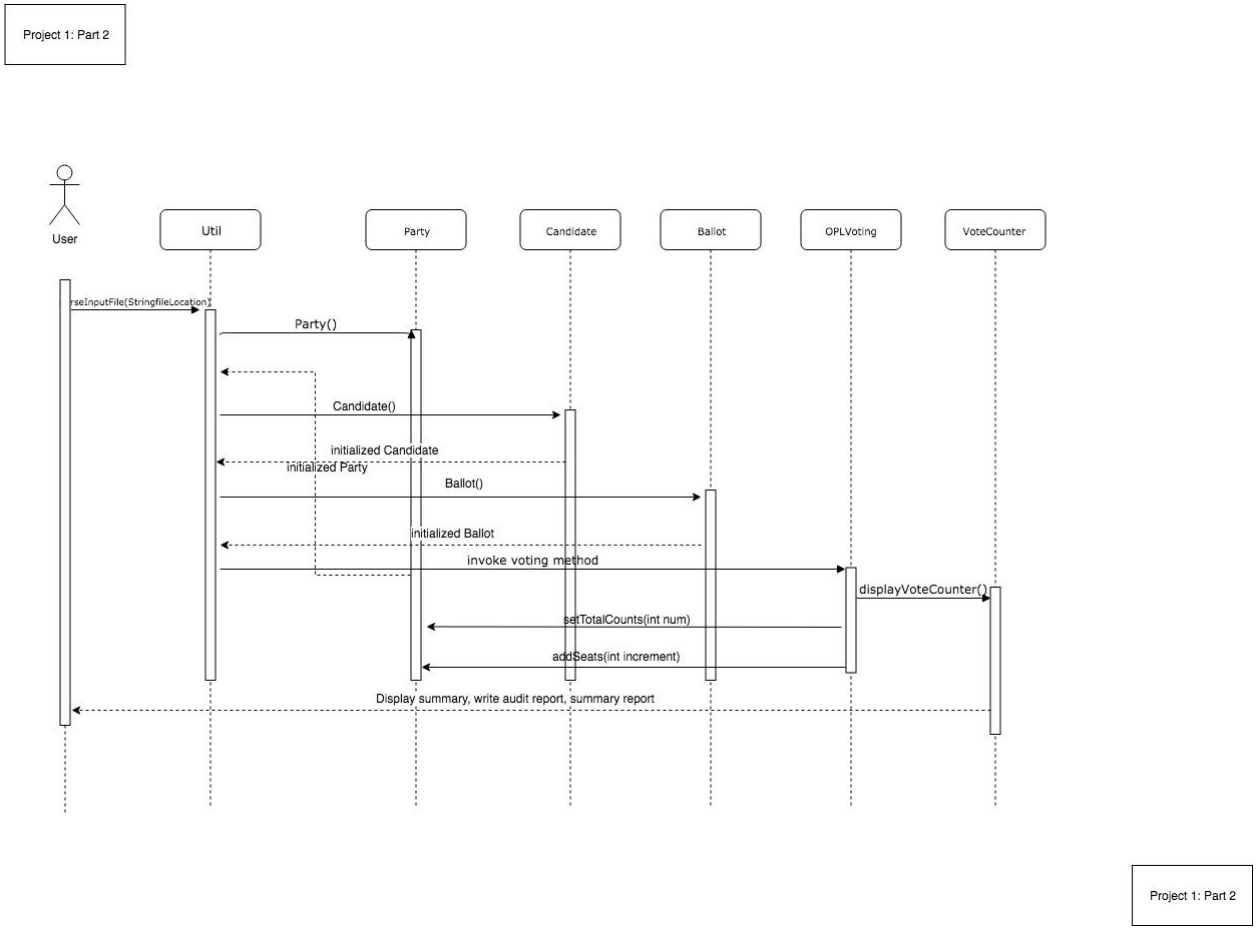
8.1.1 UML Class Diagram


<<interface>>
LogUtil
+ logger: Logger
+ makeLog(caseNum: int): void
IRVoting Class
+ candPoolIR: Candidates [0...*]
+ makeLog(caseNum: int): void
+ main(args: String[0...*]: void
+IRVoting(filename: String)
VoteCounter Class
+ logger:Logger
+ display(s: String): String
+ main(args: String[0...*]): void
OPLVoting Class
+ candPoolOPL:Candidates [0...*]
+ makeLog(caseNum: int): void
+ main(args: String[0...*]): void
+ OPLVoting(filename: String)
BufferedReader
+ readLine(): void
Util Class
N/A
parseInputFile(String fileLocation): void
Ballot Class
- id:int
-votes:int[1..*]
+Ballot(id:int, votes: int[1...*])
+ getId(): int
+ getVotes(): int[0...*]
+ setId(id: int): void
+ setVotes(votes: int[0...*])
0..*
Candidate Class
- name:String
- party:Party
- votes:int
- id:int
- status:int
- ballots:Ballot [0...*]
+ Candidate(String name, int votes)
+ reset(): void
+ getVotes(): int
+ setVotes(int votes): void
+ getName(): String
+ setName(name: int): void
+ getParty(): Party
+ setParty(party: Party): void
+ getId(); int
+ setId(id: int): void
+ getVotes(): int
+ setVotes(votes: int): void
+ getStatus(status: int): void
+ getBallots(): Ballot[0...*]
+ setBallots(ballots: Ballot[0...*]
+ addBallots(b: Ballot): void
+ addVotes(value: int): void
+ transferTo(another: Candidate):void
1
0..*
PartyClass
- cands: Candidates [0...*]
- totalCounts: int
- totalSeats: int
+ Party(int totalCounts, int seatCounts)
+ getTotalCounts(): int
+ setTotalCounts(totalCounts: int): void
+ addTotalCounts(increment: int): void
+ getSeatCounts(): int
+ setSeatCounts(num: int)
+ addSeats(increment: int)
+ getCandidates(): Candidate[0...*]
+ setCandidates(cands: Candidates[0...*]): void
+ addCandidate(cand: Candidate): void
+ toString(): String
1

Description:

The UML diagram shows the relations between different classes and interface. VoteCounter, IRVoting and OPLVoting will implement the LogUtil interface so they can report logs properly. In addition, Candidate class has direct association with Ballot class and party class. IRVoting class has association with Candidate and Ballot, and OPLVoting class has association with Ballot class, Candidate class and Party class. The system will initiate either an IRVoting instance or an OPLVoting instance to solve the given problem according to the request info in the given file.

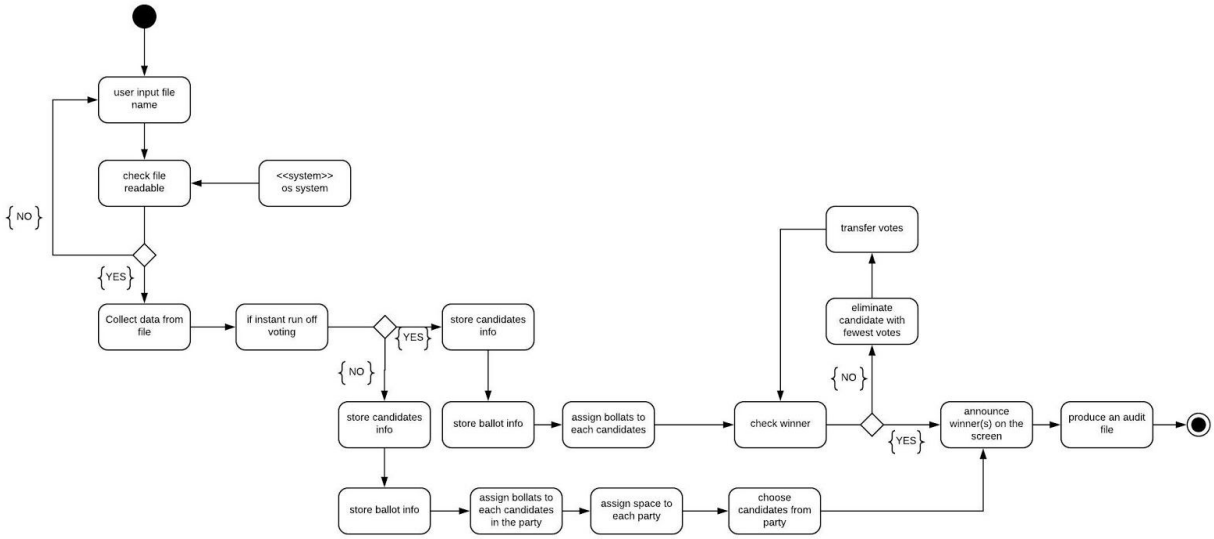
8.1.2 OPL Sequence Diagram



Description

This sequence diagram shows the process of Open Party List Voting to allocate seats. User inputs voting file to the system, then the Util class would parse the file and get related Party, Candidate and Ballot data. After the file is parsed, the system would invoke methods of OPLVoting class which would implement OPL voting process to get election result. When the election result is determined after processing all voting data, VoteCounter would display the result of the election.

8.1.3 Activity Diagram for OPL and IR



Description

The system will ask for input file until the file is valid. And it collects data from the file. If the voting information is not instant runoff system which means it is open party list system. When all the data was stored in the database, it will calculate the proportion of votes to each party and assign the total space to each party. In the end, the candidates from each party are selected. Otherwise it is instant runoff system, the system will store all the data information in the file to the database. And after that it will check the first round winner, if there is no winner, it will eliminate the candidate win fewest votes and transfer the votes until the winner appear. When the winner appears, it will display on the screen along with the information. Then an audit file will be created with the election information at the time.