

Team 11  
**Voting System**  
Software Design Document

Name (s):  
Chen Liu,  
Jicheng Zhu,  
Yingwen Weng,  
Zilong He.

Date: (02/23/2021)

## TABLE OF CONTENTS

<b><u>1. INTRODUCTION</u></b>	<b><u>2</u></b>
<u>1.1 Purpose</u>	<u>2</u>
<u>1.2 Scope</u>	<u>2</u>
<u>1.3 Overview</u>	<u>2</u>
<u>1.4 Reference Material</u>	<u>3</u>
<u>1.5 Definitions and Acronyms</u>	<u>3</u>
<b><u>2. SYSTEM OVERVIEW</u></b>	<b><u>3</u></b>
<b><u>3. SYSTEM ARCHITECTURE</u></b>	<b><u>4</u></b>
<u>3.1 Architectural Design</u>	<u>4</u>
<u>3.2 Decomposition Description</u>	<u>4</u>
<u>3.3 Design Rationale</u>	<u>11</u>
<b><u>4. DATA DESIGN</u></b>	<b><u>11</u></b>
<u>4.1 Data Description</u>	<u>11</u>
<u>4.2 Data Dictionary</u>	<u>13</u>
<b><u>5. COMPONENT DESIGN</u></b>	<b><u>13</u></b>
<b><u>6. HUMAN INTERFACE DESIGN</u></b>	<b><u>18</u></b>
<u>6.1 Overview of User Interface</u>	<u>18</u>
<u>6.2 Screen Images</u>	<u>18</u>
<u>6.3 Screen Objects and Actions</u>	<u>21</u>
<b><u>7. REQUIREMENTS MATRIX</u></b>	<b><u>21</u></b>
<b><u>8. APPENDICES</u></b>	<b><u>22</u></b>

# **1. INTRODUCTION**

## **1.1 Purpose**

This software design document describes the architecture and system design of the voting system software.

The expected audience are election officials at voting stations, who want to use our voting systems for analyzing results of two types of voting (Instant Runoff Voting, IRV & Open Party List, OPL), Programmers and testers who are interested in working on the project by further developing it to be used for more types of voting or fix existing bugs.

## **1.2 Scope**

This document contains a complete description of the design of our voting system.

The basic architecture is a voting system with the ability to handle two types of voting (instant runoff and open party list)

Team 11 members in charge of the voting system software can access to making changes when necessary. The change includes but is not limited to changing the codes of each section and fixing existing bugs.

## **1.3 Overview**

This document is divided into seven sections. We go through the introduction of the document in section 1 and overview of the system in section 2.

In section 3, we describe the system architecture. Each entity contains a description concerning its functionality.

In section 4, we focus on the data design.

In section 5, components design of the system is discussed.

In section 6, we present the human interface design.

Finally, we present the requirement matrix in section 7 and appendix.

## 1.4 Reference Material

SRS document of team 11.

Software's Github:

<https://github.umn.edu/umn-csci-5801-S21-002/repo-Team11.git>

IEEE Template for System Requirement Specification Documents:

<https://goo.gl/nsUFwy>

Instant-runoff voting, Wikipedia:

[https://en.wikipedia.org/wiki/Instant-runoff\\_voting?oldformat=true](https://en.wikipedia.org/wiki/Instant-runoff_voting?oldformat=true)

Party-list proportional voting, Wikipedia:

[https://en.wikipedia.org/wiki/Party-list\\_proportional\\_representation?oldformat=true](https://en.wikipedia.org/wiki/Party-list_proportional_representation?oldformat=true)

GUI, Wikipedia:

[https://en.wikipedia.org/wiki/Graphical\\_user\\_interface?oldformat=true](https://en.wikipedia.org/wiki/Graphical_user_interface?oldformat=true)

RAM, Wikipedia:

[https://en.wikipedia.org/wiki/Computer\\_memory?oldformat=true](https://en.wikipedia.org/wiki/Computer_memory?oldformat=true)

CSV, Wikipedia:

[https://en.wikipedia.org/wiki/Comma-separated\\_values?oldformat=true](https://en.wikipedia.org/wiki/Comma-separated_values?oldformat=true)

API, Wikipedia:

<https://en.wikipedia.org/wiki/API>

## 1.5 Definitions and Acronyms

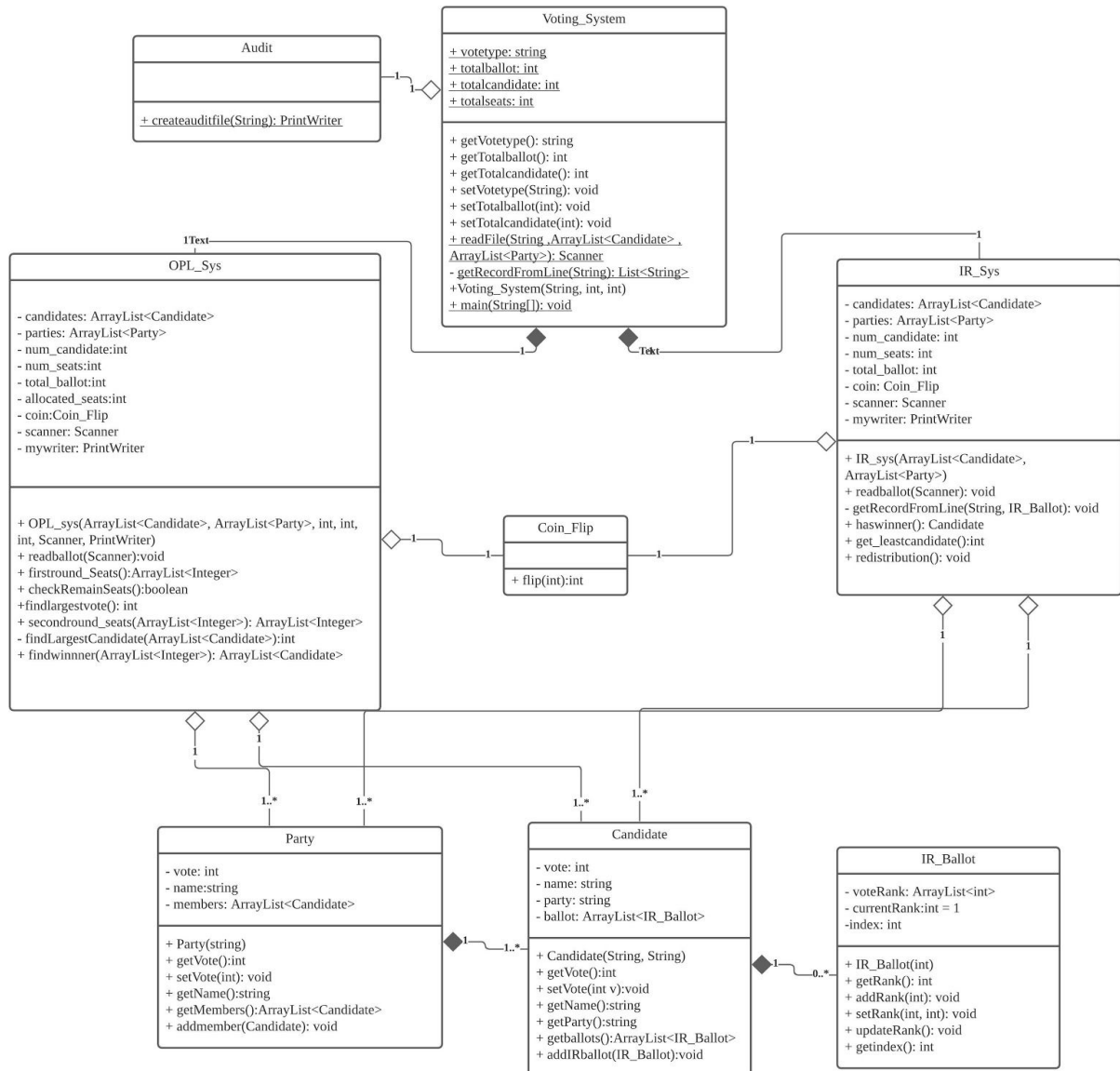
See Appendix A in SRS.

## 2. SYSTEM OVERVIEW

See section I in SRS

### 3. SYSTEM ARCHITECTURE

#### 3.1 Architectural Design



Overview: see section 3.3

#### 3.2 Decomposition Description

##### Party

Name: Party

Type: Class

Description: This is for open party list voting (OPL), this class records all information about a candidate, including his or her name, party affiliation, number of voters, and ballot list.

Attributes:     vote: int  
                  name: string  
                  members: ArrayList<Candidate>

Resources: None

Operations: New

    Name: Party

    Argument: int, string, ArrayList<Candidate>

    Returns: Void

    Pre-condition: candidate and party name is read as a string

    Post-condition: create a new party object

    Exceptions: None

    Flow of Events:

- 1, pass initial int, string and Party into
- 2, return by creating a new party object

Operations: get value

    Name: getVote()

    Argument: None

    Returns: int

    Pre-condition: None

    Post-condition: the number of voters of the candidate is returned

    Exceptions: None

    Flow of Events:

- 1, return the number of voters of the candidate

    Name: getName()

    Argument: None

    Returns: string

    Pre-condition: None

    Post-condition: the name of candidates is returned

    Exceptions: None

    Flow of Events:

- 1, return the name of the candidate

    Name: getParty()

Argument: None  
Returns: string  
Pre-condition: None  
Post-condition: the party of the candidate is returned  
Exceptions: None  
Flow of Events:  
    1, return the party of the candidate

### **Candidate**

Name: Candidate

Type: Class

Description: This is for open party list voting (OPL), this class records all information about a candidate, including his or her name, party affiliation, number of voters, and ballot list.

Attributes:     vote: int  
                  name: string  
                  party: string  
                  ballot: ArrayList<Ballot>

Resources: None

Operations:

    Name: Candidate()  
    Argument: string, Party  
    Returns: None  
    Pre-condition: candidate and Party name is read as a string  
    Post-condition: create a new candidate  
    Exceptions: None  
    Flow of Events:  
        1, pass string and Party into  
        2, return by creating a new candidate

Operations: get value

    Name: getVote()  
    Argument: None  
    Returns: int  
    Pre-condition:  
    Post-condition: the number of voters of the candidate is returned  
    Exceptions: None  
    Flow of Events:

1, return the number of voters of the candidate

Name: getName()

Argument: None

Returns: string

Pre-condition: None

Post-condition: the name of candidates is returned

Exceptions: None

Flow of Events:

1, return the name of the candidate

Name: getParty()

Argument: None

Returns: string

Pre-condition: None

Post-condition: the party of the candidate is returned

Exceptions: None

Flow of Events:

1, return the party of the candidate

### **Voting system**

Name: Voting\_System

Type: class

Descriptions: Parent class for OPL\_Voting\_Sys and IR\_Voting\_Sys. Store the general information including total ballot and total candidate. Determine the type of vote by reading the first line of the file. Ballot counting and other processing are done by the respective child class.

Attributes:     votetype: string  
                  totalballot: int  
                  totalcandidate: int

Resources:None

Operations: get value

    Name: getvotetype()

        Arguments: None

        Returns: string

    Pre-condition: Successfully read the ballot file and the type of vote is stored in the attribute votetype.

    Post-condition: Vote type is returned.

    Exceptions: None

    Flow of Events:

        1, Return attribute votetype.



Name: getTotalballot()

Arguments: None

Returns: int

Pre-condition: Successfully read the ballot file and the number of ballots is stored in the attribute totalballot.

Post-condition: The number of ballots is returned

Exceptions: None

Flow of Events:

- 1, Return attribute totalballot.

Name: getTotalcandidate()

Arguments: None

Return: int

Pre-condition: Successfully read the ballot file and the number of candidates is stored in the attribute totalcandidate.

Post-condition: The number of candidates is returned.

Exceptions: None

Flow of Events:

- 1, Return attribute totalcandidate.

Name:readFile(string)

Arguments: string

Return: FileInputStream

Pre-condition: file name is read as a string

Post-condition: The content of the file is stored in a file stream. useful information is stored in respective attributes.

Exceptions: None

Flow of Events:

- 1, Use the string that contains filename to read the file and stored in a file stream.
- 2, The first line determines the type of vote, and stores it in the attribute votetype.
- 3, The second line has the number of candidates, store it in the attribute totalcandidate.
- 4, If it is IR, go to the fourth line and store the number in the attribute totalballot.  
If it is OPL, go to the fifth line and store the number in the attribute totalballot.

### **OPL voting system**

Name: OPL\_Voting\_Sys

Type: class

Description: The child class of Voting\_System. Main class for performing OPL counting.

Attributes: candidates: ArrayList<Candidate>

parities: ArrayList<Party>

Operations: file processing

Name: processfile(FileInputStream)

Arguments: FileInputStream

Returns: None

Pre-condition: The type of vote is OPL

Post-condition: File finish processing and the winner(s) is declared.

Exceptions: None

Flow of Events: (see details in the OPL activity diagram)

- 1, Read the file.
- 2, get the first line of the file and determine the vote type.
- 3, get the 2nd line for the number of candidates
- 4, get the 3rd line for name and party of candidates
- 5, get the 4th line for number of seats
- 6, get the 5th line for total number of ballots (N)
- 7, read the next line and get ballot information
- 8, increase the ballot count for the party & increase the ballot count for the candidate
- 9, check if reach end of file. If no, reach to step 7. If yes, end of file processing

## **IR\_Ballot**

Name: IR\_Ballot

Type: class

Descriptions: A specific class to record the data of the ballot. For each ballot, it contains a different rank of vote.

Attributes:     voteRank:ArrayList<int>  
                  currentRank:int

Resources:None

Operations: get value

Name: IR\_Ballot()

Arguments: None

Returns: None

Pre-condition: None

Post-condition: current rank is set to one.

Exceptions: None

Flow of Events:

- 1, Return IR\_Ballot object.

Name: getRank()

Arguments: None

Returns: int

Pre-condition: Successfully read the ballot file and the vote rank for each ballot is stored in the object class.

Post-condition: None

Exceptions: None

Flow of Events:

- 1, Return attribute currentRank.

Name: updateRank()

Arguments: None

Returns: void

Pre-condition: IR\_Ballot object is initialized.

Post-condition:  $\text{currentRank} = \text{currentRank} + 1$

Exceptions: None

Flow of Events:

- 1, Increase currentRank by 1
- 2, Return None.

## **IR\_Voting\_Sys**

Name: IR\_Voting\_Sys

Type: class

Descriptions: The child class of Voting\_System. Main class for performing IR runoff ballot.

Attributes: candidates: ArrayList<Candidate>  
parties: ArrayList<Party>

Resources: None

Operations: file processing

Name: processfile()

Arguments: FileInputStream

Returns: None

Pre-condition: The type of vote is IR Ballot.

Post-condition: File finish processing and the winner(s) is declared.

Exceptions: None

Flow of Events(see details in the IR activity diagram):

- 1, Read the file.
- 2, get the first line of the file and determine the vote type.
- 3, get the 2nd line for the number of candidates

- 4, get the 3rd line for name and party of candidates
- 5, get the 4th line for total number of ballots (N)
- 6, read the next line and create a IR\_Ballot
- 7, assign the IR\_Ballot to the first priority candidate
- 8, check if reach end of file. If no, reach to step 6. If yes, end of file processing

### **3.3 Design Rationale**

We plan to initialize 7 classes to achieve this project. OPL\_Voting\_Sys and IR\_Voting\_Sys are two classes that extend from Voting\_System. Voting\_Sys has the general information about the election, OPL\_Voting\_Sys and IR\_Voting\_Sys store the respective information for two types of votes. We have a Coin\_Flip class to help with the tie condition.

Party class stores the general information for a party. It has an integer to store the total vote, a string to store the party name, a candidate list to store the candidates within a specific party.

In our approach, we create an IR\_Ballot class to store the ballot information. By the same time, we also add the ballot to each Candidate object. Once we need to eliminate the least voted candidate, we only need to process the vote from the least candidate and get the ballot object information of the next candidate vote to reassign the ballot to it. We believe this approach is more efficient than the other architecture.

Another architecture uses ballots to store the candidate information. If we want to redistribute the ballot we will need to go through the entire ballot list again to find the next candidate. This method is inefficient since our ballot list can be very large.

## **4. DATA DESIGN**

### **4.1 Data Description**

**Party**

Attribute name	Attribute type
vote	Int
name	string
Members	ArrayList<Candidate>

Vote: the number of votes a party has

Name: the name of the party

Members: a list of candidates belongs to this party

### **Candidate**

Attribute name	Attribute type
vote	Int
name	string
party	string
ballot	ArrayList<IR_Ballot>

Vote: the number of votes a candidate has

Name: the name of the candidates

Party: the name of the party this candidate belongs to.

Ballot: a list of ballots belongs to this candidate, only used for IR

### **Voting\_System**

Attribute name	Attribute type
votetype	string
totalballot	int
totalcandidate	int

Votetype: the type of vote, can be "OPL" or "IR".

Totalballot: the total number of ballots in the file

Totalcandidate: the total number of candidates in the file

**OPL\_Voting\_Sys**

Attribute name	Attribute type
candidates	ArrayList<Candidate>
parties	ArrayList<Party>

Candidates: a list of candidates participate in this vote

Parties: a list of parties participate in this vote

**IR\_Voting\_Sys**

Attribute name	Attribute type
candidates	ArrayList<Candidate>
parties	ArrayList<Party>

Candidates: a list of candidates participate in this vote

Parties: a list of parties participate in this vote

**IR\_Ballot**

Attribute name	Attribute type
voteRank	ArrayList<int>
currentRank	int

VoteRank: the order of ranks in this ballot

CurrentRank: the current rank of this ballot, starts from 1 and increases 1 for every redistribution.

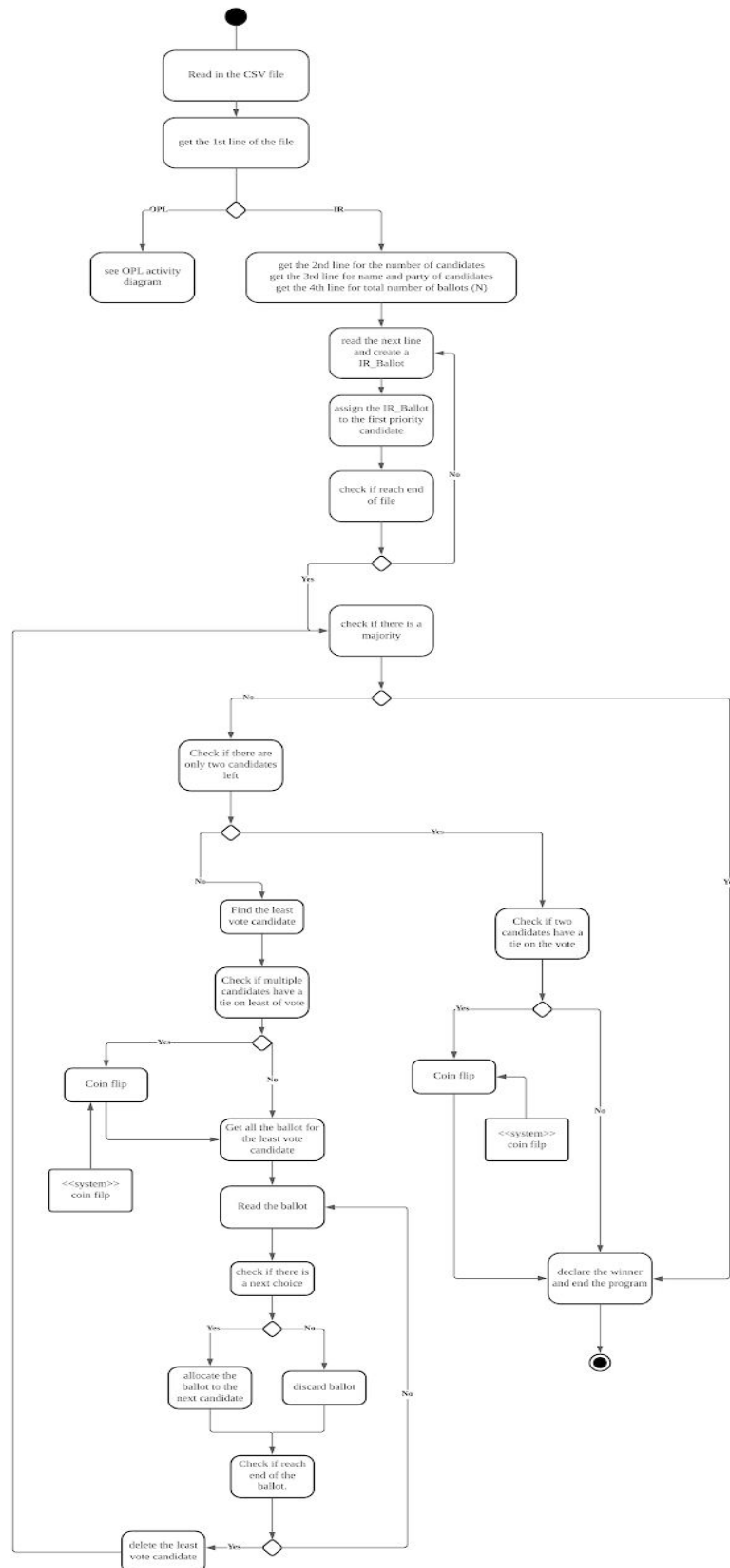
**4.2 Data Dictionary**

See Section 3.2 and 4.1.

**5. COMPONENT DESIGN****5.1 IR Activity Diagram**

See document [IR\\_activity\\_diagram\\_Team11.pdf](#) for a clear diagram.

IR Overview: Start by reading the first line of the CSV file, start counting if OPL, jump to IR voting if not. Get the voting information from lines 2 through 4, and then create IR\_Ballot objects for each ballot. [2]Check if you reach the majority. If yes, declare the winner and end of the program, if no, check if there are only two candidates left. In this case, if yes, check if two candidates have a tie. If yes, do a coin flip and declare the winner. Otherwise, declare the winner. If there are more than two candidates left, check if there is a tie on the lowest vote. If no, get all the ballots for the least voted candidate. If yes, do a coin flip, then get all the ballots for the least vote candidate. After this, get all the ballots of lowest candidates and [1]read them. Check if there is a next choice. If yes, allocate the ballot. If not, discard the ballot. Check if reached the end of the least candidates ballot. If no, return to [1]. If yes, delete the least voted candidate and return to [2].

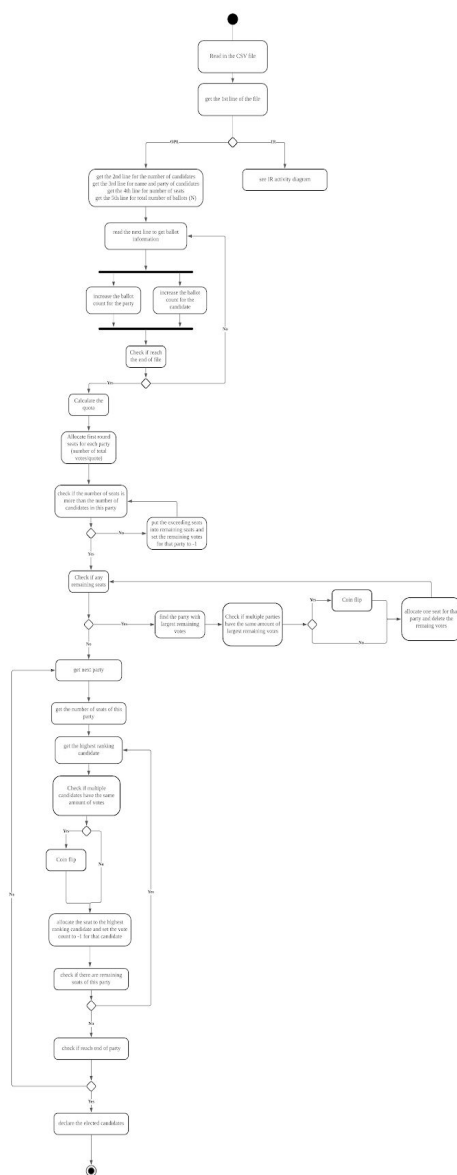




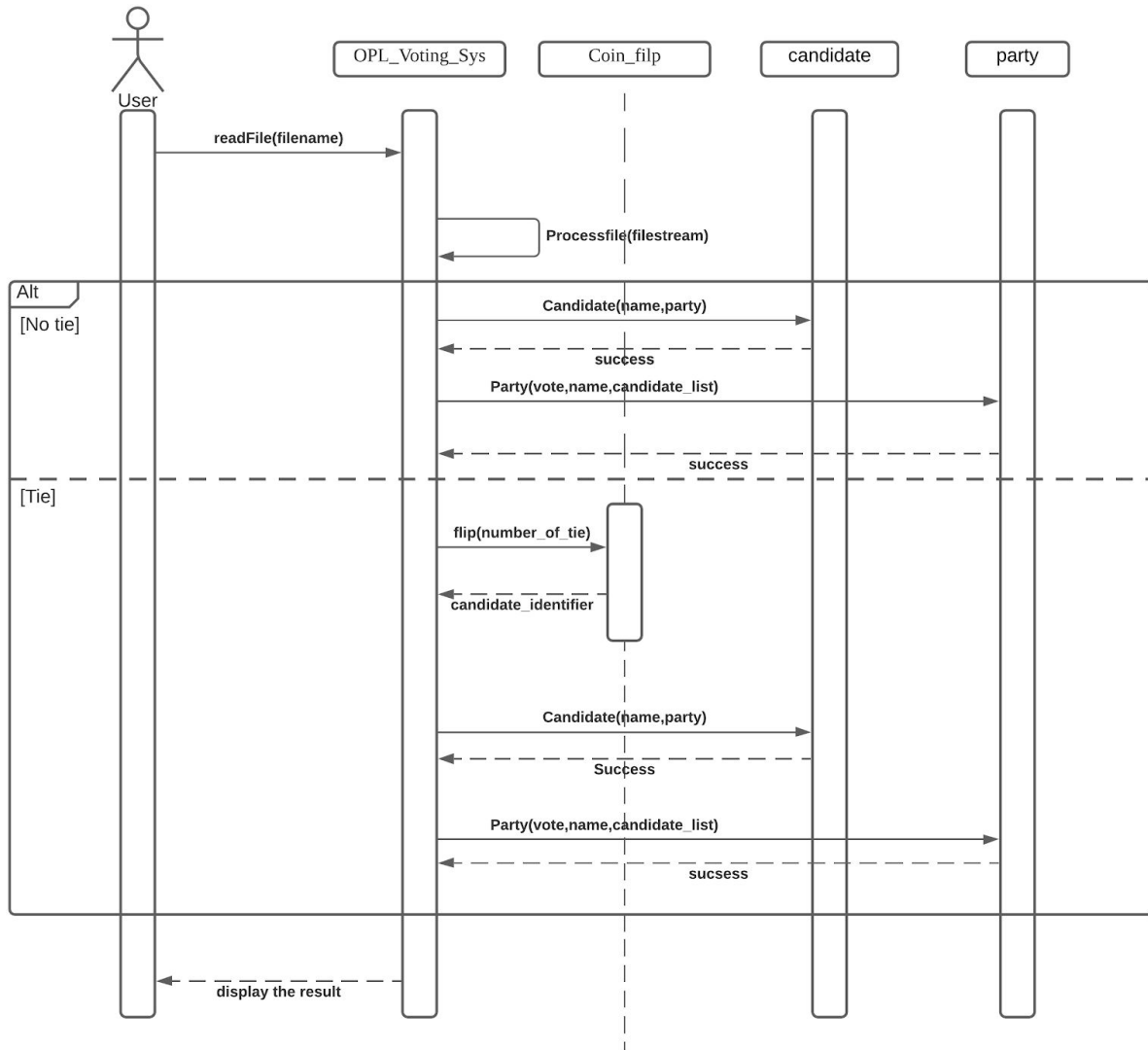
## 5.2 OPL Activity Diagram

See document [OPL activity diagram Team11.pdf](#) for a clear diagram.

OPL Overview: Start by reading the first line of the CSV file, start counting if OPL, jump to IR voting if not. Get the voting information from lines 2 through 5, and then increase the vote count for both parties and candidates. Calculate the quota for each party, if one party's seat number is greater than its candidate number, distribute the extra seats to the other party. Finish quotas and allocate seats to party candidates. For any tie in the vote count, a coin flip is used to settle the tie.



### 5.3 OPL Sequence Diagram



- 1, The user triggers the `readFile()` method in a `OPL_Voting_Sys` object class, supplying the name of the file. `OPL_Voting_Sys` is the main class which does the file processing and vote count.
- 2, The OPL voting system calls the `Processfile(filestream)` method to allocate the ballots to the corresponding parties and candidates.
- 3, If there is a tie, do a coin flip and return a candidate identifier. Then allocate the seats to the candidates.
- 4, If there is no tie, the seats are allocated to the candidates based on ranking.

5, The result is returned from OPL\_Voting\_Sys class.

## **6. HUMAN INTERFACE DESIGN**

### **6.1 Overview of User Interface**

Our system will have the following seven functionality.

Read: read the input file.

Decision: determine the vote type.

Process 1: File processing for Instant Runoff.

Process 2: File processing for Open Party Runoff.

Two-side Decision: Coin Flip simulation.

Display: Display the results on the screen for media.

Audit: Generalize the auditing file.

1. Users will provide the file of voting data as an input file. This will make the system satisfy the read functionality.
2. Users will choose a voting type in the provided input file. This will make the system complete the decision functionality.
3. The system will process the file based on the user defined requirements. (either process 1 IR or process 2 OPL)
4. The feedback information will be displayed on the screen. Also, users can access the information and procedure through the audit file. It records each step the system does to the data.

### **6.2 Screen Images**

This program works for Windows, Mac and Linux, but we only provide the brief interface of the mac environment. Users need to adjust the command based on their system.

```
XXX@XXX ~: cd WaterFall_Project
```

```
XXX@XXX ~/WaterFall_Project: javac Voting_Sys.java  
input_1.csv
```

```
XXX@XXX ~/WaterFall_Project: java Voting_Sys
```

Counting is over!

Type: IR

Number of ballot: 100000

Kleinberg (R): 52000

Rosen (D): 18000

Chou (I): 30000

The winning elector:

Kleinberg (R)

Thank for using the vote counting system.

```
XXX@XXX ~/WaterFall_Project: ls
```

```
Voting_Sys.java  Voting_Sys  input1.csv  audit.txt
```

This briefly shows the interface of IR ballot. The example of the audit.txt file is provided in the SRS document section 3.1.3.

XXX@XXX ~/WaterFall\_Project : javac Voting\_Sys.java  
input\_2.csv

XXX@XXX ~/WaterFall\_Project : java Voting\_Sys

Counting is over!

Type: OPL

Number of Ballot: 100

Number of Seats: 4

Parties:

D: 50%, 50%, 2 Seats

R: 31%, 31%, 1 seats

I: 19%, 19%, 1 seats

Candidates:

Pike(D): 30

Foster(D): 20

Deutsch(R): 10

Borg(R): 20

Jones(R): 1

Smith(I): 19

The winning electors

Pike(D)

Foster(D)

Borg(R)

Smith(I)

Thank you for using the vote counting system.

This briefly shows the situation of the Open Party List ballot. The example of audit.txt file is provided in the SRS document section 3.1.3.

### 6.3 Screen Objects and Actions

On our screen, the only option is for the user to provide the input file. We don't have a GUI for users right now. There are no more actions. Details are listed in section 3.1 of the SRS document.

## 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.

Use cases ID	use cases name	Class	Functions
VT_001	Read the input file	voting_system	readFile() getTotalballot() getgetTotalcandidate()
VT_002	Determine the vote type	voting_system	getVotetype()
VT_003	File processing for IR	IR_Ballot IR_Voting_sys	IR-Ballot() getRank() UpdateRank() Processfile()
VT_004	File processing for OPL	Candidate Party OPL_Voting_Sys	Candidate(string, Party) Candidate::getVote() Candidate::getName() Candidate::getParty() Party(int, string, ArrayList<Candidate>) Party:: getVote() Party:: getName() Party:: getMembers() Processfile()

VT_005	Coin flipping simulation	Coin_Flip	flip(int):int
VT_006	Display the result to the screen	IR_Voting_sys OPL_Voting_Sys	Processfile()
VT_007	Generate the Audit file	IR_Voting_sys OPL_Voting_Sys	Processfile()

## 8. APPENDICES

*not applicable*