

Team SASS
IR/OPL Voting Software
Software Design Document

By: Samuel Wong(wong0613), Andy Chen(chen6640), Sai
Tallapragada(talla037), and Sree Pemma(pemma003)

Table of contents

Table of contents	2
1. Introduction	3
1.1 Purpose	3
1.2 Scope	3
1.3 Overview	3
1.4 Definitions and Acronyms	4
2. System Overview	4
3. System Architecture	4
3.1 Architectural Design	4
3.2 Decomposition Description	5
3.3 Design Rationale	5
4. Data Design	6
4.1 Data Description	6
4.2 Data Dictionary	6
5. Component Design	7
6. Human Design Interface	10
6.1 Overview of User Interface	10
6.2 Screen Images	11
7. Requirements Matrix	12

1.Introduction

1.1. Purpose

This software Design Document provides the design details of the IR/OPL voting software.

The intended audience for this document is for election officials looking to use the software and programmers/developers/testers looking to better understand the software and its structure.

1.2. Scope

The IR/OPL voting system is a tool to assist election officials in determining the results of an election. Officials can use the software to easily process and generate an audit file containing a summary and the results of the election from an election information data file that's passed into the software, and display the results as well. The software is capable of running 100,000 ballots in under 8 minutes and is programmed to resolve any ties should one arise.

1.3. Overview

This SDD Document provides information important to understanding the structure of the IR/OPL voting system. This includes detailed descriptions and diagrams of classes and how they interact with each other during execution of the program(Section 3), relevant activity diagrams for the program(Section 3), description of data design/data structures(Section 4), and a brief description of the pgrams Human Interface design(Section 6).

1.4. Definitions and Acronyms

- Throughout the document the Instant Runoff and Open Party Listing Voting software will be referred to as the IR/OPL Voting software
- IR: Instant Runoff
- OPL: Open Party Listing

2. System Overview

Our system is capable of handling both IR and OPL voting. It is designed in such a way that it takes in the file name which contains the ballots for all candidates in the IR/OPL election. The system then counts the ballots and assigns/redistributes the votes based on the type of election to declare a winner. If a tie needs to be broken, the necessary steps to do so will be taken. An audit file and a statistical summary including the winner of the election will be generated at the very end.

3. System Architecture

3.1. Architectural Design

For a high level overview of the subsystems of the program and their activities see the following diagrams:

- **IR_Activity_Diagram_Team24.jpg**
- **OPL_Activity_Diagram_Team24.jpg**

These diagrams give a high level description of how the program works and how the responsibilities of the system are split between different objects/classes. Each of these diagrams begins with a prompt for filename input which happens in the main class. This main class isn't shown in either diagram but contains the object for the election being run, relevant information needed to create the election object obtained from the input file, and the necessary methods to obtain this information from the file.

3.2. Decomposition Description

For a detailed description of the decomposition of the objects/classes contained in the system see the following diagram:

- **UML_Class_Diagram_Team24.jpg**

Also see the following sequence diagram to see the interactions/flow of events between the user of the system, the system, and the OPL Election class during the execution of an OPL Election:

- **OPL_Sequence_Diagram_Team24.jpg**

3.3. Design Rationale

While designing the above program architecture there were several important issues and trade/offs that were considered.

First, we ran into the issue of how the program would deal with the Ballot information. For an IR election the ballots needed to be stored so that they could be redistributed as needed. However, for an OPL election each ballot should only have to be touched by the program once. This gave rise to the issue of whether or not we needed to store the Ballots in their own class or not for both elections. We decided to store the Ballot information in a Ballot object for both election types as this information would later be needed to produce an audit file as well. Storing this information would be more efficient than processing the file again to track where each Ballot was going.

Second, was the issue of where the brunt of the work was going to occur for the program. Originally we thought we could simply have the program's current Election class act as a Main class for the program and have this class doing most of the work. However we then saw it beneficial to have an actual Main class to handle some of the preconditions that must be met for the Election class to work properly. Having a separate Main class allows the program to handle taking user input for the filename and creating a File object that can then be passed into an Election class as a parameter. This approach was taken in order to keep activities irrelevant to the running of the election outside of the Election class.

Lastly, we decided that we needed to differentiate between two different types of Ballots, IR Ballots and OPL Ballots, rather than group them into the same class. This was due to the slight differences in the voting schemes. It would have been possible to group these together, but

this would lead to a small amount of irrelevant information being stored in the Ballot object in the case of either election being run.

4.Data Design

4.1. Data Description

The data starts in the form of a CSV file. The IR/OPL system will read in the CSV file upon user input. The file contents being read will then be stored in an Election object.

4.2. Data Dictionary

Object	Attributes and Types	Methods and parameters
Ballot	ID: int	getID(): int setID(int): void
Candidate	votes: Ballot[]	getBallots(): Ballot[] getNumVotes(): int
Election	ballots: Ballot[] candidates: Candidate[] file: File totalVotes: int	assignBallot(Ballot,Candidate): void breakTie(Candidate[]): Candidate createAuditFile(): File createMediaFile(): File processBallots(File): Ballot[]
IR Ballot	candidates: Candidate[]	getHighestVote(): String
IR Election	candidates: Candidate[]	determineWinner(): Candidate eliminateCandidate(): void readCandidates(File): Candidate[] redistributeVotes(Candidate): void
Main	election: Election electionFile: File electionType: String	createElection(String,File): Election displayInfo(): void getElectionType(): String getFileName(): String main(String[]): void

		quit(): void
OPL Ballot	candidate: Candidate	getCandidate(): String
OPL Election	candidates: Candidate[] parties: Party[] seats: int	createsParties(): Party[] distributeSeats(): void getQuota(): int readCandidates(File): Candidate[]
Party	candidates: Candidate[] numSeats: int	getNumVotes(): int getTopCandidates(int): Candidate[]

5. Component Design

Main Class

main(String[]):void

- The main function in the Main class will be what is running the program. When the User starts running the program, the main() will prompt the user to input a csv file containing election information.
- Then the main function will open the File object and read the file.
- When we reach the line which contains Election information. The main() will call on createElection(String, File) where String is type of election and File is the file object we are reading through.
- displayInfo() will be called next.
- quit() is finally called at the end and this will quit the system

createElection(String,File):Election

- This will call on the main() of either OPL Election class or IR Election Class depending on the String that is passed in which indicates the type of election.
- We will access the OPL Election class or IR Election class by first instantiating objects of those classes. Both the OPL and the IR Election classes will have a global variable which will store the File Object we are reading so we will also pass our file object FILE into these classes. After doing that, we will finally call on the main() of the OPL Election class or IR Election class.

displayInfo():void

- This function will display the end results of either the IR/OPL election to the screen which includes winners, distribution of seats, number of ballots cast, and stats for all the candidates.

getElectionType(): String

- Return election type using string.

getFileName(): String

- Gets the file object we are using to read the input csv file

quit(): void

- Will quit the system after the statistical summary and audit file has been created

Election Class

assignBallot(Ballot, Candidate):void

- assignBallot(Candidate) will assign

breakTie(Candidate[]): Candidate

- If there is a tie between two candidates for a number of votes, this function will be called and we will use this function to break the tie and declare one of the candidates to be ranked above the other.

createAuditFile(): File

- Creates an audit file which contains the statistical summary of the election regarding winners and the election results for all the parties and candidates.

createMediaFile(): File

- Creates a media file which contains the winner and the number of votes they have

processBallots(File): Ballot[]

- This function will take in the file and read the ballots and look for a candidate object in the global candidate array which corresponds to the name of the person on the ballot
- Then, it will pass in this candidate object into the assignBallot(Ballot, Candidate) which will assign the current Ballot to the appropriate candidate object

OPL Election Class

main():void

- The first thing that will be done in this main function is to call the readCandidates() function
- The processBallots() function will be called next to assigned each ballot to its respective candidate
- Then createParties() is called on to create a global array of party object
- Finally, main will call on distributeSeats()

getQuota():int

- Returns the quota for the election

readCandidates(File):Candidate[]

- Will read the file information about candidates and create a Candidate class object for each candidate and we will store all these objects in a global variable Candidate Array.

distributeSeats():void

- This function will distribute seats to the parties in this election.

createParties():Party[]

- This function will create a global array of party objects based on the parties of each candidate that has been read in the readCandidates() function.
- The global array will be parsed in order to make sure no party is duplicated in this array.
- This function will also add all of the candidate objects(Candidates themselves) to their parties after we create each party.

IR Election Class

main(): void

- Call readCandidates() to read in the Candidates
- Call processBallots() to assign ballots to Candidates
- Repeatedly loop determineWinner() until a winner is determined

determineWinner():Candidate

- This is called once all the ballots have been read in
- Iterate through all the candidates and see if one has more than 50% of the votes.
- If there is one, then return the Candidate as the winner.
- Otherwise call eliminateCandidate() to eliminate the candidate with fewest first choice votes

eliminateCandidate():void

- This is called once all the ballots have been read in
- Iterate through all the candidates and determine the candidate with the fewest first choice votes
- If there is a tie, call breakTie() to pick a Candidate fairly and randomly
- Call redistributeVotes() with on that Candidate to take that Candidate's votes and redistribute them to the other Candidates in the array
- Delete that Candidate from the array

readCandidates(File):Candidate[]

- Read in the candidates from the third line of the file
- Create an array of the Candidates listed and return it

redistributeVotes(Candidate):void

- Take Candidate being passed as the argument and redistribute their votes to their next choice by going through the Ballot[] of that Candidate

Candidate Class

getBallots():Ballot[]

- This function is in charge of returning all the ballots for the candidate object that is associated with the candidate that we are looking at.

getNumVotes():int

- This function returns the number of votes that a specific candidate has

Ballot Class

getID():int

- Returns the ID of the ballot which is an int

setID(int):void

- Sets the ID to a ballot

OPL Ballot

getCandidate():String

- Returns the name (a string) of the candidate to which the ballot is casted to

IR Ballot

getHighestVote():String

- Returns the name (a string) of the candidate who has the highest number of votes in that particular ballot

Party Class

getNumVotes():int

- This function returns the number of votes that have been casted to that party and all the candidates in it

getTopCandidates(int):Candidate[]

- Returns an array of candidate objects in that party that are the top number (number of candidates as the int param specifies)

6. Human Design Interface

Overview of User Interface

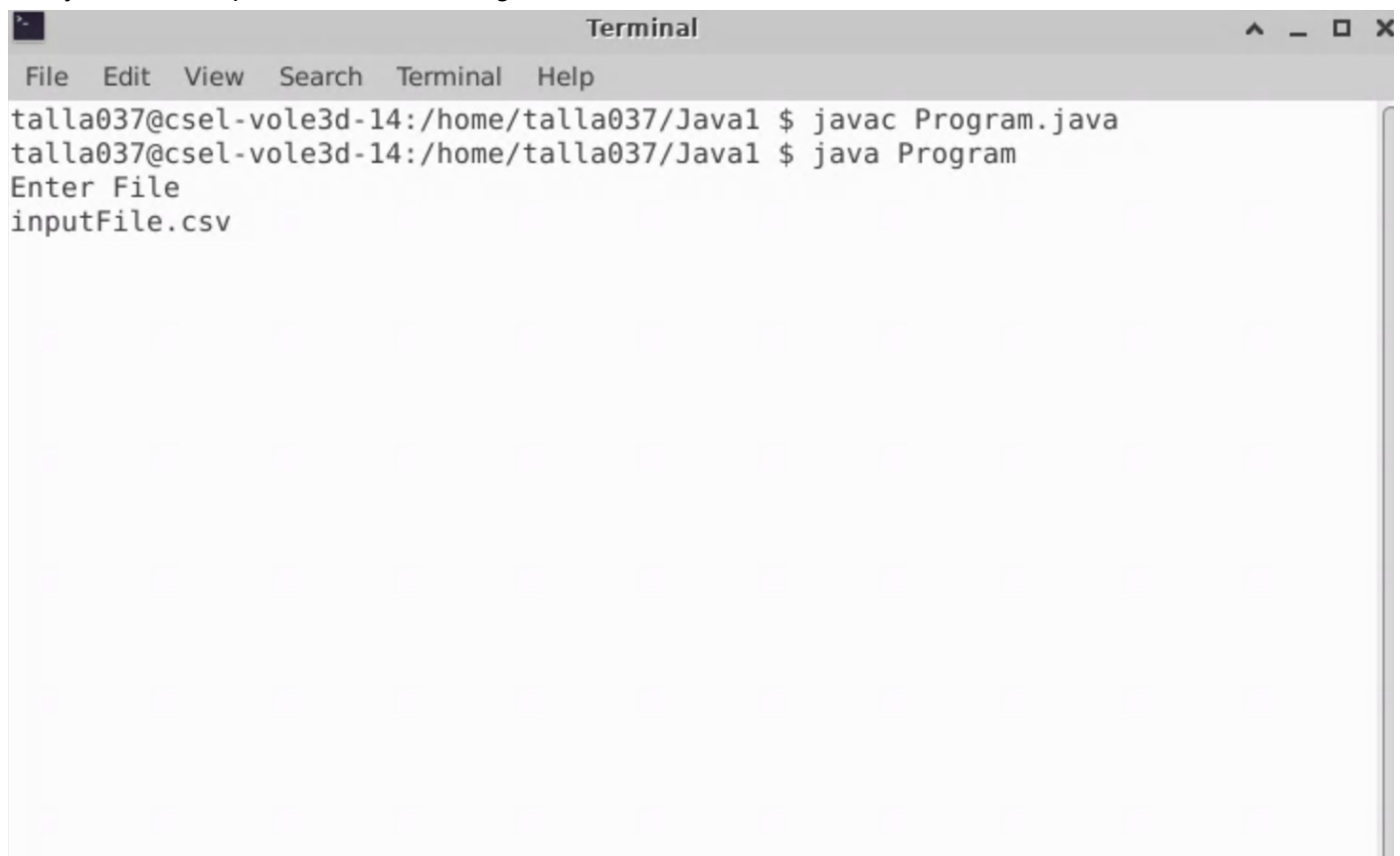
(Note: Need a computer for this)

The User will be using a terminal to run our program. The user will first open up the terminal and then go to the appropriate directory of where the java program is located. Then the user will compile and run the java program.

The User will be prompted to enter a csv file by the program and the user will enter the path to the input csv file which contains election information. Then the program will process the election results in the background as the user waits. Once the program is done, it will let the user know that an audit file is produced which will have the statistical summary of the election Results for the user to view. The user can view this file by opening up the file system and going to the src folder located in the directory this java program is running in.

Screen Images

The Screen shot below shows how a user can compile the Program and run the program and finally enter the input csv file containing the election information.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the user "talla037@cse1-vole3d-14" in the directory "/home/talla037/Java1". The user enters the command "javac Program.java", followed by "java Program". The program prompts "Enter File" and the user enters "inputFile.csv".

```
Terminal
File Edit View Search Terminal Help
talla037@cse1-vole3d-14:/home/talla037/Java1 $ javac Program.java
talla037@cse1-vole3d-14:/home/talla037/Java1 $ java Program
Enter File
inputFile.csv
```

7. Requirements Matrix

Use Case Number	Implementation in Design
UC_001	In our design, we have a Main class which has a main() function. This function will prompt the user to pass in a file name which will be opened afterwards.
UC_002	We will open the file and start reading the file in the main() of the Main class.
UC_003	The main() function in the main class will get the type of election and it will call on the createElection() function where the main() of the IR election class will start running if we are supposed to run that type of an election. The createElection() will instantiate an object of the IR Election class if this is the type of election we are supposed to be running.
UC_004	The main() function in the main class will get the type of election and it will call on the createElection() function where the main() of the OPL election class will start running if we are supposed to run that type of an election. Even more, the createElection() will instantiate an object of the OPL Election class if this is the type of election we are supposed to be running.
UC_005	When running the IR election, if we come across a tie between two candidates, the Election class has a function called breakTie() and we can call this function from the IR Election class to break a tie between two candidates by ranking one of the candidates above the other one.
UC_006	The Main class has a displayInfo() function which will get called at the end.
UC_007	The Election class has a createAuditFile() function that handles this use case.
UC_008	Statistical summary is generated within the createAuditFile() function before the audit file is created.

UC_009	In our Election Class, we have processBallot() and this function will read through the file for ballot information. As it comes across each ballot, this function will also assign a unique ID to the ballot.
UC_010	In our Election Class, we have processBallot() and this function will read through the file for ballot information. After assigning an ID to each ballot in this function, we will call on the function assignBallot() which will assign each ballot to its corresponding candidate. Finally, if you want to receive the number of ballots casted for any of the candidate class, we have function called getNumVotes() in the Candidate Class and if you access to the candidate class, you can call on getNumVotes() to get the number ballots casted for a candidate.