Software Design Document

for

Voting System™

Version 1.0 approved

Prepared by Daniel Rockcastle (rockc004), John Caspers (caspe158), Michael Birk (birkx031), and Mukesh Pragaram Choudhary (choud108)

CSCI 5801 Team 6

October 28, 2018

Table of Contents

1. Introduction

1.1. Purpose

This Software Design Document describes the architecture and system design of our voting system, including the design of both the open party list election system and the instant runoff voting system.

1.2. Scope

This document describes the first implementation of the Voting System™. The system has two major functions. First, it should determine the winner of an open party ballot election and create an audit file for it. Secondly, it should be able to determine the winner of an instant runoff election and create an audit file for it. This document will not go into testing of the software.

1.3. Overview

This document will be divided into several sections. First, we have the introduction. In section two, there will be a general overview of the system. In section three, the specific architecture will be outlined. In section four, the data design will be outlined. In section five, the component design will be outlined. Section six will outline the human interface design, while section seven will cross reference our SRS document.

1.4. Definitions and Acronyms

OPL -> Open Party List
IRV -> Instant Runoff Voting

2. System Overview

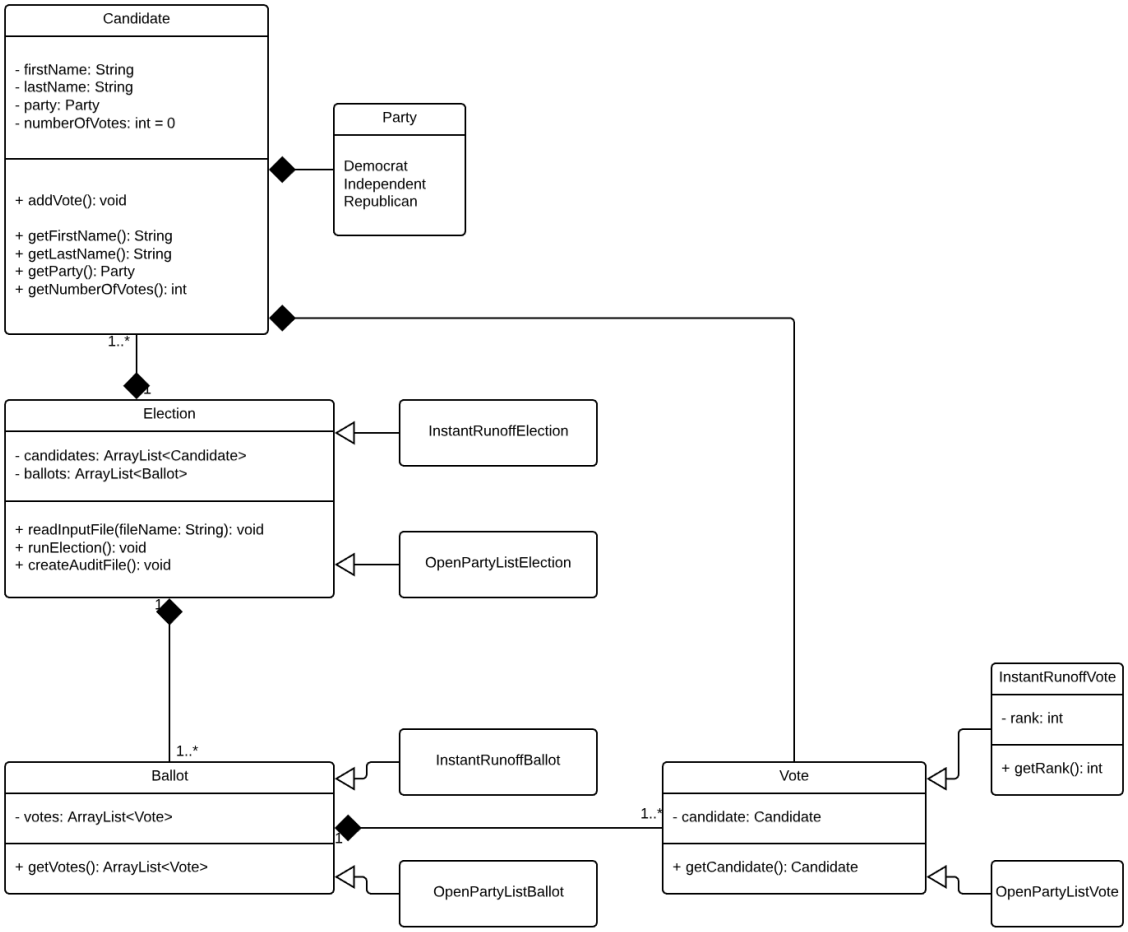
As mentioned in the introduction, our Voting System™ has been designed to accommodate two different styles of election voting. The first of these styles is called Instant Runoff Voting, or IRV for short. This system of voting has voters rank candidates on the ballot. At each run-through of the votes, the candidate with the lowest number of first place votes is eliminated. Their first-place votes then go to the voter's second place candidate. The election ends in the event that one candidate has received over 50% of the voting. The second style, known as Open Party List, or IPL for short, has voters vote for individual candidates. These individual candidate votes are assigned to a party, who then wins an amount of seats proportional to the amount of votes they received. The actual candidates who win the election are decided by ranking the candidates per party.

The system design will be outlined in more detail in the following sections. However, a basic overview is as follows. For IRV, all voter rankings will first be recorded from the file. At every step, the candidate with the least number of first place votes will be eliminated. These votes will be allocated to the voter's second place preference, unless their second-place preference is already removed in which case they will be allocated further down the list. Votes will continue to be reallocated until a candidate reaches above 50% of the total votes. They will then be declared

the winner. For OPL, all votes will first be counted and assigned to a party. A number of seats won per party will then be assigned based on the number of votes they proportionally received. Candidates will also be ranked per party in this step. Any remainder seats won will then be added. Finally, actual winners will be determined by taking the top N candidates per party, where N is the number of seats the party won. All steps for both systems will also be recorded in an audit file to ensure nothing fishy occurred.

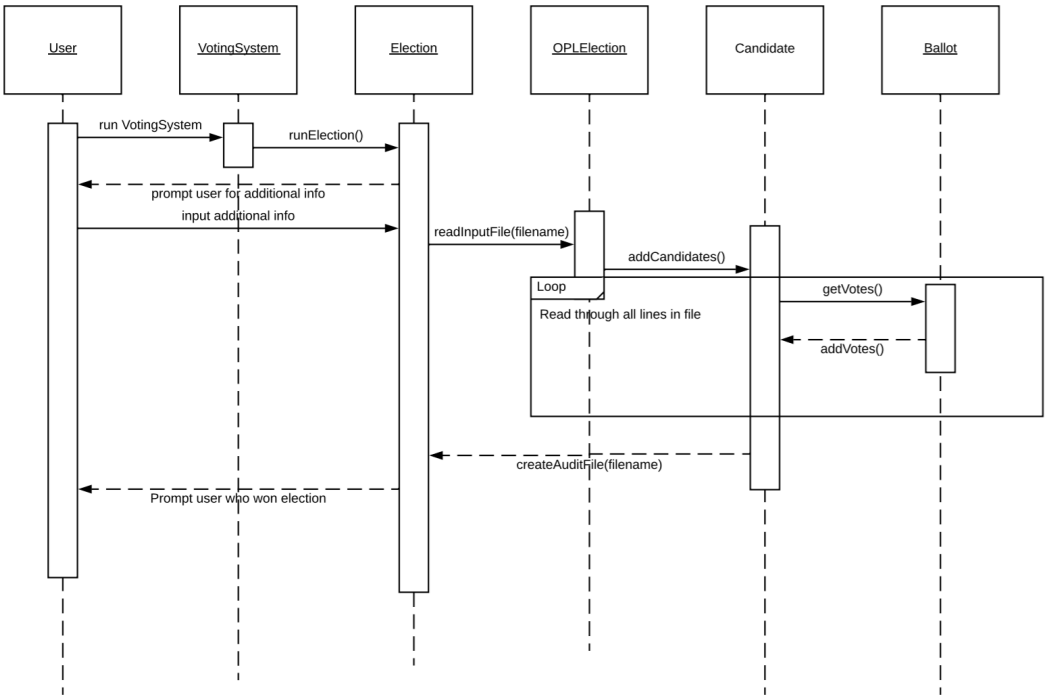
3. System Architecture

3.1. Architectural Design



Overview of class diagram for Voting System™. Specifies the objects in the system and their attributes. `Election` is the central class, and it is an interface; `InstantRunoffElection` and `OpenPartyElection` both implement it. `Candidate` specifies the candidates in the election and their characteristics. `Party` is an enum specifying all possible parties that a candidate can belong to. `Ballot` is an abstract class and represents the individual ballots used in the election; `InstantRunoffBallot` and `OpenPartyBallot` extend it. `Vote` is an abstract class and represents each individual vote on each ballot, and it contains a candidate. `InstantRunoffVote` and `OpenPartyListVote` extend it, and `InstantRunoffVote` adds a rank attribute.

3.2. Decomposition Description



Overview of sequence diagram for OPL. The user must run the voting system program with an input file that contains the type of election, candidates, and all votes. The system will then prompt the user for any additional information necessary. The program will then read in the input file and begin to parse through it. First, it will determine the type of election that will take place. In this diagram, Open Party List Election voting will take place, so the program will run that type of election. Next, the candidates will be determined from the third line of the file. Then the program will start to read each vote line which will determine who the voter voted for and add that vote for that candidate. Once all votes are added to the candidates, the winner will be determined and an audit file with the results will be created. The user will then be prompted on who won.

4. Data Design

4.1. Data Description

The main data of this system are candidates and ballots. Candidates are represented as a list of `Candidate` objects in the main `Election` class, and ballots are represented as a list of `Ballot` objects in the same class. A `Candidate` object contains a first name and last name (represented as `String` objects), a party (all possible parties are listed in a `Party` enum), and the number of votes the candidate currently has (represented as an `int`). A `Ballot` object simply contains a list of votes contained on the ballot. Votes are represented by `Vote` objects; `Vote` objects contain a `Candidate` and, in the case of an instant runoff election, also contain a rank (represented as an `int`). The candidate and ballot lists are initialized by parsing a given input file. This data is then processed by

iterating through the list of ballots and adding the appropriate votes to each instance of `Candidate`.

4.2. Data Dictionary

Objects:

- `Ballot`
  - Attributes:
    - `votes`
      - Type: `ArrayList<Vote>`
      - Description: list of votes contained on the ballot
  - Methods:
    - `getVotes`
      - Type: `ArrayList<Vote>`
      - Description: getter for `votes` attribute
- `Candidate`
  - Attributes:
    - `firstName`
      - Type: `String`
      - Description: first name of the candidate
    - `lastName`
      - Type: `String`
      - Description: last name of the candidate
    - `numberOfVotes`
      - Type: `int`
      - Description: number of votes that the candidate currently has; is constantly updated as the program runs
    - `party`
      - Type: `Party`
      - Description: party of the candidate
  - Methods:
    - `addVote`
      - Type: `void`
      - Description: iterates `numberOfVotes` attribute by 1
    - `getFirstName`
      - Type: `String`
      - Description: getter for `firstName` attribute
    - `getLastName`
      - Type: `String`
      - Description: getter for `lastName` attribute
    - `getNumberOfVotes`
      - Type: `int`
      - Description: getter for `numberOfVotes` attribute
    - `getParty`
      - Type: `Party`

Description: getter for `party` attribute

`Election`

Attributes:

`candidates`

Type: `ArrayList<Candidate>`

Description: list of candidates in the election

`ballots`

Type: `ArrayList<Ballot>`

Description: list of ballots in the election

Methods:

`createAuditFile`

Type: `void`

Description: creates the audit file and writes the appropriate election information to it

`readInputFile`

Type: `void`

Description: parses the given input file into appropriate candidate and ballot objects

Parameters:

`fileName`

Type: `String`

Description: name of the input file to be read and parsed

`runElection`

Type: `void`

Description: calculates the results of the election by iterating through the list of ballots

`Vote`

Attributes:

`candidate`

Type: `Candidate`

Description: the candidate this vote corresponds to

`rank` (instant runoff voting only)

Type: `int`

Description: the rank of the candidate this vote corresponds to (possible range: 1 to number of candidates)

Methods:

`getCandidate`

Type: `Candidate`

Description: getter for `candidate` attribute

`getRank` (instant runoff voting only)

Type: `int`

Description: getter for `rank` attribute

5. Component Design

The objects that are a part of the voting system are Election objects, Candidate objects, Ballot objects, and vote objects. An Election object can either be for an Instant Runoff Election or an Open Party List Election. An Election will contain the candidates and ballots with a way to read the input file and create an audit file. A Candidate object will have a first name (String), last name (String), party (String), and number of votes (Integer). A Ballot can either be for IRV or OPL which will contain votes. A vote will be associated with a specific candidate. The way the vote is interpreted depends on if the election is OPL or IRV. An audit file contains the information of how the election went and who won.

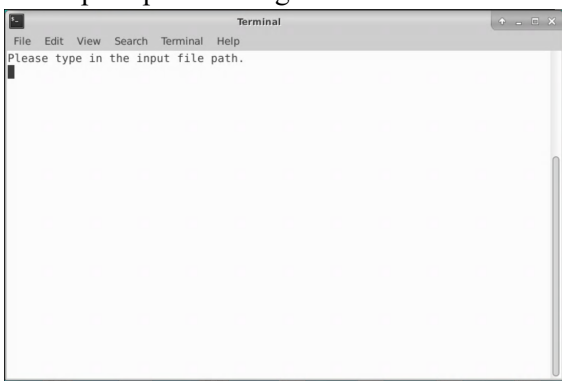
6. Human Interface Design

6.1. Overview of Human Interface

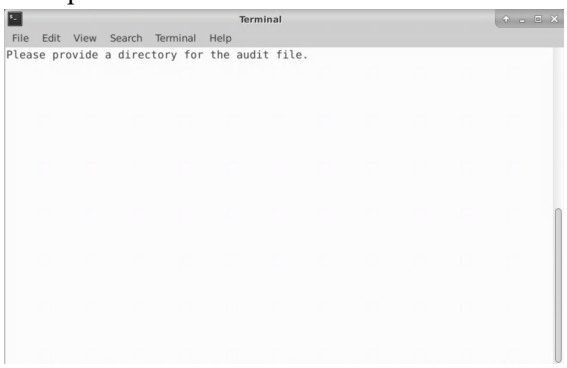
The user can either run this program through the terminal or a GUI. In each case, the user can provide an input file when they initially run it, or they will be prompted to provide such file. The program will parse the file and run the appropriate election. After the program finishes running, the winner of the election will be displayed on screen and an audit file will be created containing the results which can be viewed by the user.

6.2. Screen Images

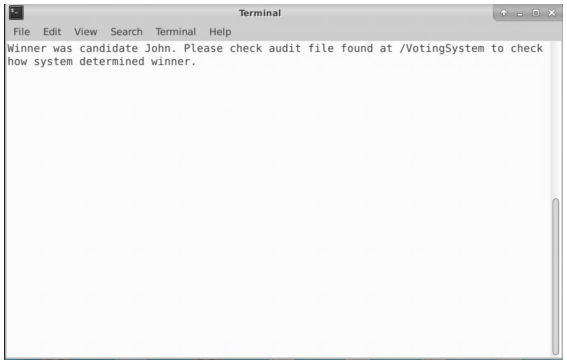
Initial prompt for voting file:



Prompt for location of audit file:



Winner displayed:



6.3. Screen Objects and Actions

We project that there will be two prompts in the terminal window associated with running the program. The first of these prompts will simply ask the user to provide an input file path where their ballot is located. The second will ask them to provide a directory for their audit file be outputted in. Since the system will ensure a unique audit file name, the user must only provide a directory. After the winner of the election is determined, the system will output the winner and the location and name of the audit file for that election so that the user can check on it.

7. Requirements Matrix

SRS 4.1: Election Official Requests Audit File -> createAuditFile(String fileName) from section 4. Also will use function readInputFile(String fileName) to read the input file for the election. All functions/objects/methods in section 4 will ultimately be used to create the audit file since it effectively tests the whole system, but createAuditFile(String fileName) is the closest parallel to it.

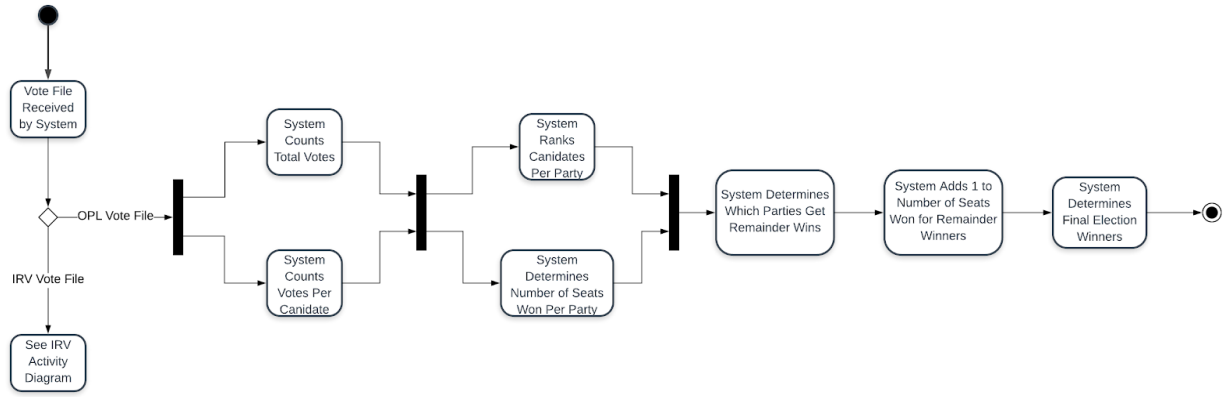
SRS 4.2: Tester Requests Audit File - Instant Runoff -> createAuditFile(String fileName) from section 4. Also will use function readInputFile(String fileName) to read the input file for the election. All functions/objects/methods in section 4 will ultimately be used to create the audit file since it effectively tests the whole system, but createAuditFile(String fileName) is the closest parallel to it. This will use specific parts of the methods designed to run the Instant Runoff voting method.

SRS 4.3: Tester Requests Audit File - Open Party -> createAuditFile(String fileName) from section 4. Also will use function readInputFile(String fileName) to read the input file for the election. All functions/objects/methods in section 4 will ultimately be used to create the audit file since it effectively tests the whole system, but createAuditFile(String fileName) is the closest parallel to it. This will use specific parts of the methods designed to run the Open Party voting method.

SRS 4.4: User Views Election Results -> This will be displayed on the terminal after the election is done running. Please look at section 6.2 for images of the terminal output when the election is done running. Once again, this will use all functions and methods found above since it is a full run-through of the program.

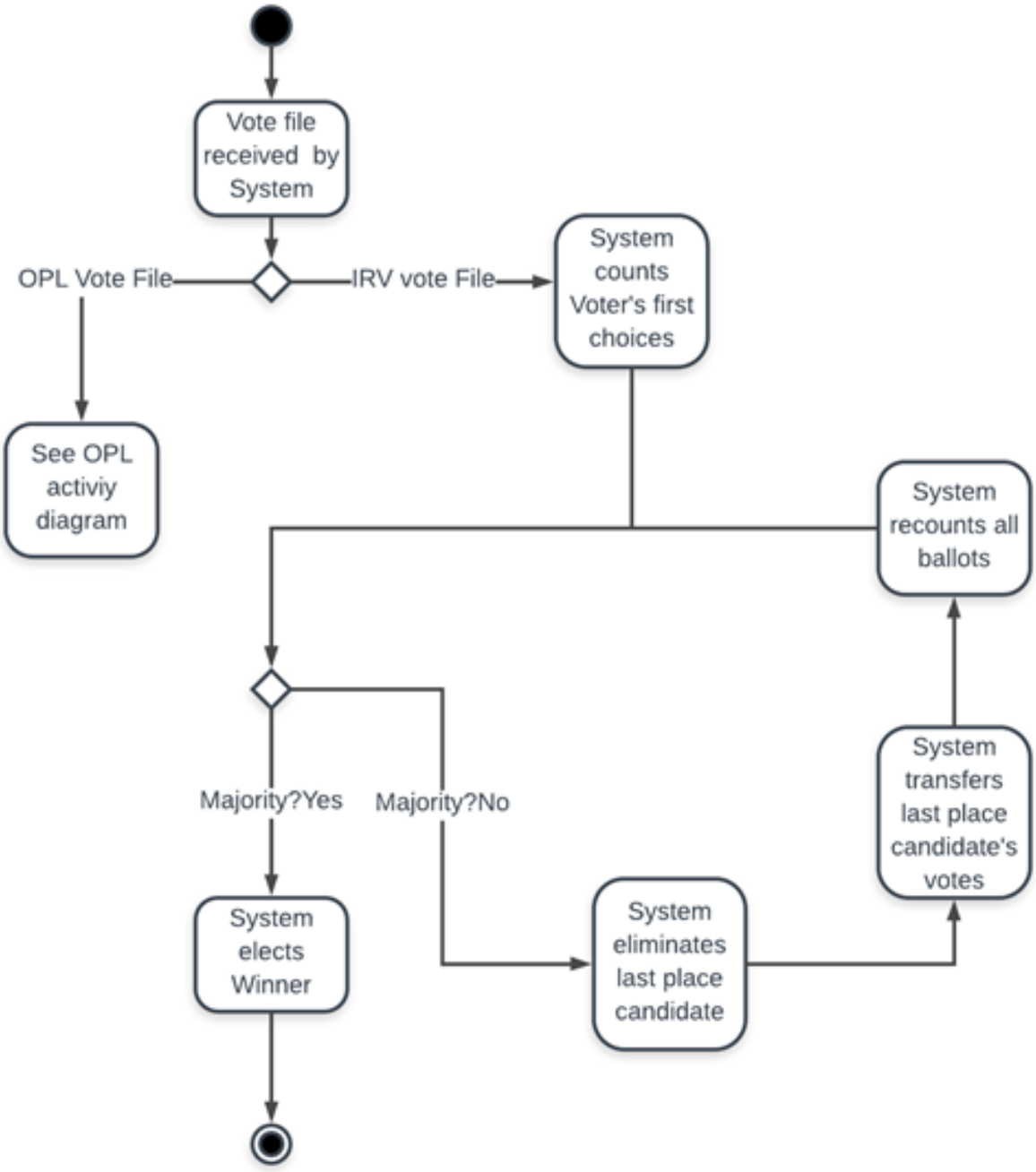
SRS 4.5: User Views Audit File -> This will be done using some type of text editor. The audit file should be openable in a text editor and able to be read in plaintext.

Activity Diagram for Open Party List Voting



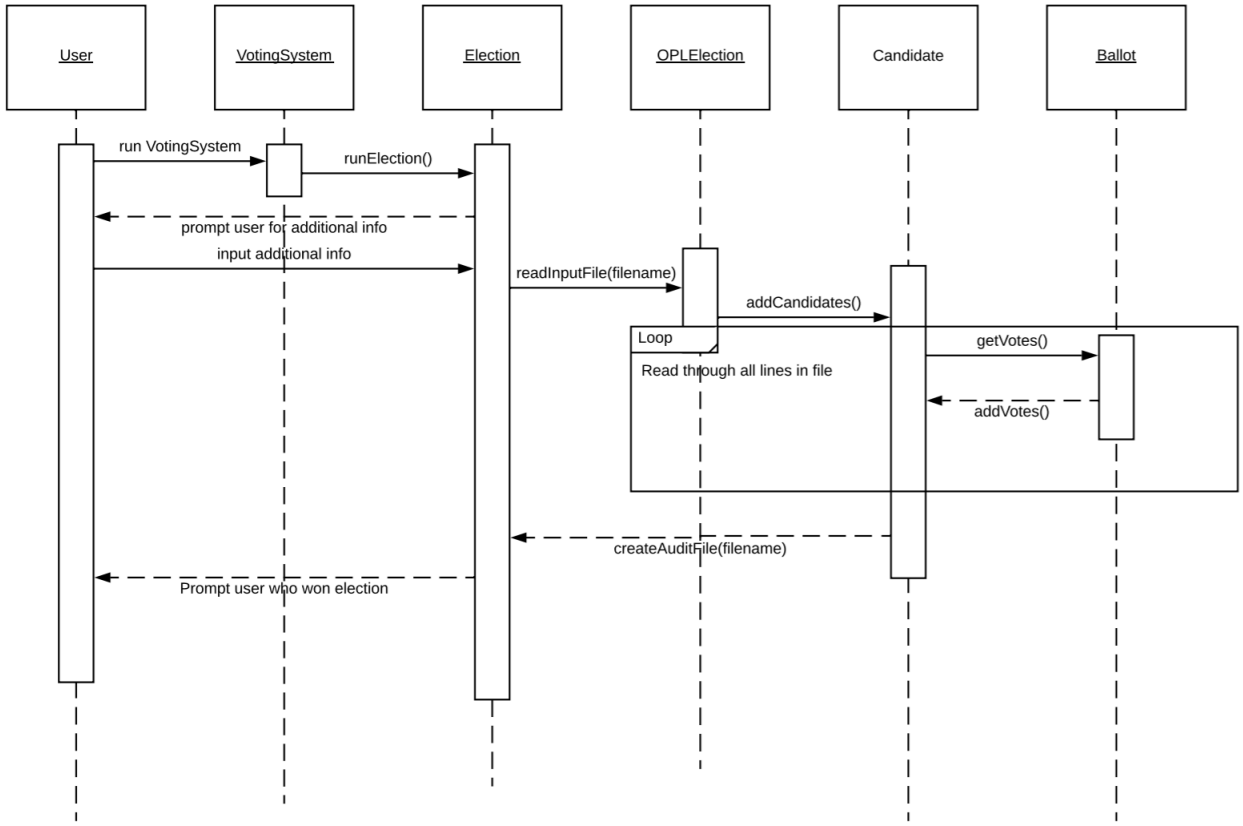
In this activity diagram, the system first opens a vote file that is given to it. It then checks to see if it is an OPL vote file or an IRV vote file. In the event that it is an OPL vote file, then we continue the process on this activity diagram. In the event that it is an IRV file, we should look at the IRV activity diagram instead. After determining that the file is an OPL vote file, the system first must both count the total number of votes and the number of votes per candidate. The total number of votes will be used to determine the number of votes required to win a single seat (total votes/num seats). The votes per candidate will be used to determine the ranking of candidates per party. After counting the total number of votes and the number of votes per party in any order, the system can continue to the next step. In this step, it both ranks the candidates in their respective parties and also determines the number of seats won per party. In order to determine the number of seats per party, the system simply takes the number of votes a party received and divides it by the number of votes required to win a single seat. This will give them the initial number of seats won. After ranking the candidates and determining the number of seats won per party in any order, the system will then use the remainder method outlined in the documentation to allocate remaining seats. It will then add one won seat to any parties that win a seat using the remainder method. After doing this, the system will determine the actual winners by taking the ranking of candidates per party and selecting the top N candidates, where N is the number of seats the party won.

Activity Diagram for Instant Runoff Voting



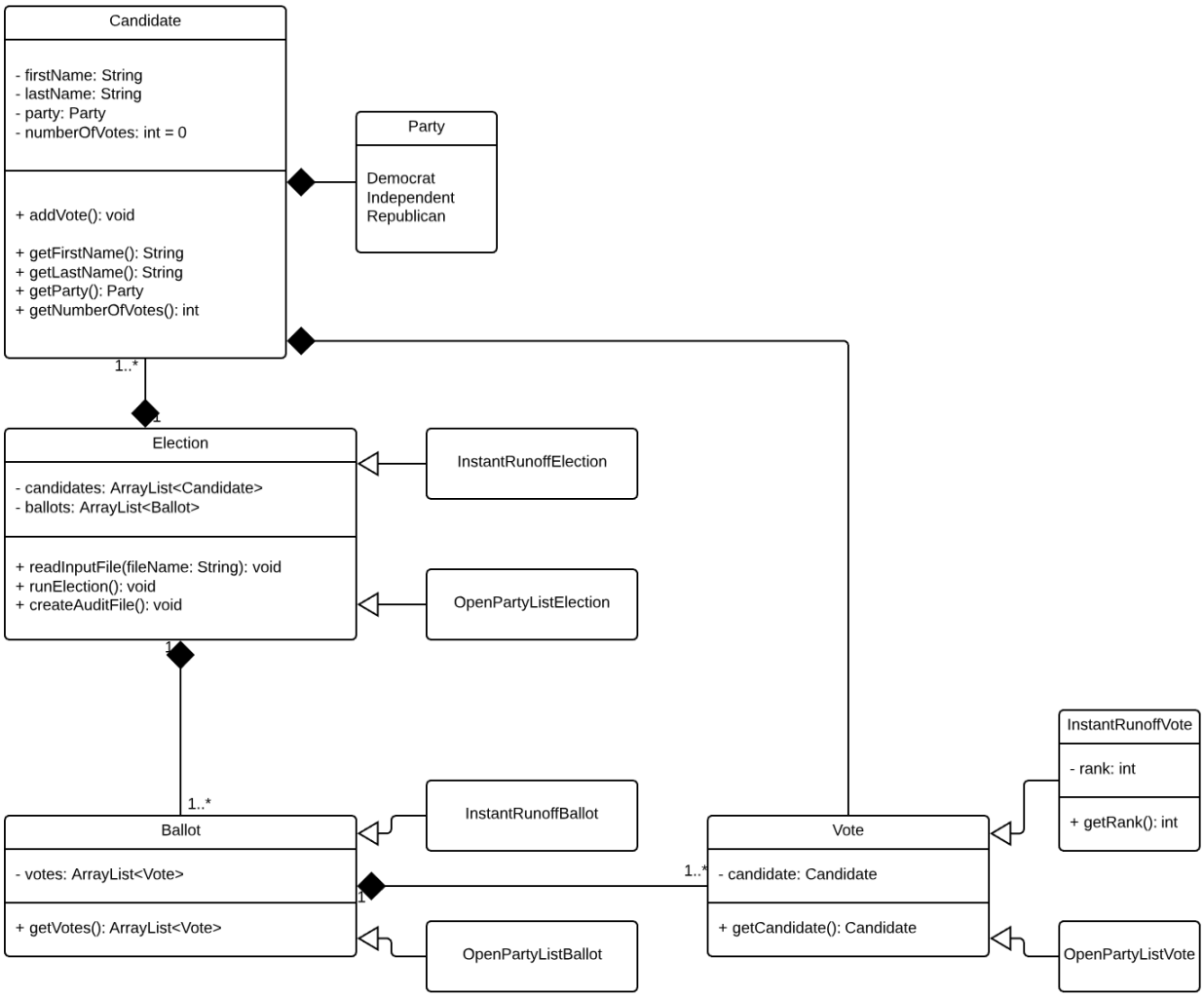
In this activity diagram, the system first opens a vote file that is given to it. It then checks to see if it is an OPL vote file or an IRV vote file. In the event that it is an IRV vote file, then we continue the process on this activity diagram. In the event that it is an OPL file, we should look at the OPL activity diagram instead. After determining that the file is an IRV vote file, the system first counts the first preference of the voters. If there is a majority then the candidate with the majority votes is elected. If not, then the system eliminates the last place candidate and transfer his votes to the remaining candidates depending on the preference. Then the ballots are re-counted by the system and this goes on until one person has a majority and he's declared the winner.

Sequence Diagram for Open Party List Voting



The user must run the voting system program with an input file that contains the type of election, candidates, and all votes. The system will then prompt the user for any additional information necessary. The program will then read in the input file and begin to parse through it. First, it will determine the type of election that will take place. In this diagram, Open Party List Election voting will take place, so the program will run that type of election. Next, the candidates will be determined from the third line of the file. Then the program will start to read each vote line which will determine who the voter voted for and add that vote for that candidate. Once all votes are added to the candidates, the winner will be determined and an audit file with the results will be created. The user will then be prompted on who won.

Class Diagram for Voting System™



Overview of class diagram for Voting System™. Specifies the objects in the system and their attributes. `Election` is the central class, and it is an interface; `InstantRunoffElection` and `OpenPartyElection` both implement it. `Candidate` specifies the candidates in the election and their characteristics. `Party` is an enum specifying all possible parties that a candidate can belong to. `Ballot` is an abstract class and represents the individual ballots used in the election; `InstantRunoffBallot` and `OpenPartyBallot` extend it. `Vote` is an abstract class and represents each individual vote on each ballot, and it contains a candidate. `InstantRunoffVote` and `OpenPartyListVote` extend it, and `InstantRunoffVote` adds a rank attribute.