Team 20

Voting Implementation
Software Design Document

Names: Danial Syed (syed0053), Mohamed Mohamed(moha1113), Omavi Collison(coll1396), Nabeel Azam(azamx016)

Date: (03/03/2023)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This SDD (Software Design Document) serves as a comprehensive overview of the software architecture and design for the polling algorithm used in elections. The document acts as a guide for our development in order to implement the system according to the design specifications. This paper is aimed at stakeholders who are interested in the software architecture and design of the voting algorithm, as well as testers, election officials, and others. The system's functional and non-functional criteria, design limitations, design methodology, and system architecture are all thoroughly outlined.

1.2 Scope

The software's main objective is to compile and sift through all of the given data (ballots) and determine which candidate has won the election. This decision is based on the voting algorithm (either closed party listing or instant runoff) and the candidates along with the number of votes they have received which is all given by the input ballot file. Once a winner is determined, the results are shared in several different ways such as receiving just the winner or a number of statistics that are made available based on the individual's role wanting to view them. With this software, compilation of ballots and redistribution of voting results will allow for a candidate to emerge victorious in a fair, unbiased manner.

1.3 Overview

This document exists to preface the voting algorithmic process we will develop. We have organized this document in a manner that outlines what our software will accomplish by diving deep into design and usage of our software by our consumers. Our reasoning behind using a certain algorithm will be highlighted in section 3. Specifics such as data usage and pseudocode will be delivered in sections 4 and 5 respectively. How the software will actually function and be used is highlighted in section 6. Furthermore, the background and context of our environment and system constraints are provided in section 2.

1.4 Reference Material

Project Github:

https://github.umn.edu/umn-csci-5801-01-S23/repo-Team20

1.5 Definitions and Acronyms

Algorithm: a process or set of rules to be followed in calculations or other problem-solving operations.

Terminal: A display that allows for execution of a program

CSV (Comma Separated Files): A type of text file where the data is separated line by line via a comma

Git: A set of tools that allows code to be transferred locally to an external repository

GitHub: The landing platform for our code and acts as the above mentioned external repository

Environment: An area that withholds all tools needed to run code such compilers and other associated developmental tools

Compiler: A translator that changes programming language code (such as Java) into the lowest form (machine code) which allows the computer to understand and output what has been programmed

IR: a ranked preferential voting method. It uses a majority voting rule in single-winner elections where there are more than two candidates.

CPL (closed p): A voting system in which voters can effectively only vote for political parties as a whole.

Command line interface: A text-based interface that allows users to interact with a program using commands entered through the command line.

Use Case: A description of how a user interacts with a system to achieve a specific goal or objective.

Matrix: A table or grid that represents a set of data in rows and columns.

SRS (Software Requirements Document): A document that outlines the functional and nonfunctional requirements of a software system. This was previously created for this software and can be found on the above linked GitHub repository.

Modular Design: A design approach that breaks a system down into smaller, independent components or modules, each with a specific responsibility.

Codebase: The collection of source code that makes up a software system.

Debugging: The process of identifying and fixing errors or defects in software.

Subsystems: Smaller, independent components or modules within a larger system, each with a specific responsibility.

Architecture: The overall design and structure of a software system, including its components, interactions, and dependencies.

Modifications: Changes made to the source code of a software system to add new functionality, fix errors, or improve performance.

Inheritance: A programming concept that allows a new class to be based on an existing class, inheriting its attributes and methods.

Composition: A programming concept that allows classes to be combined and composed to form more complex objects.

2. SYSTEM OVERVIEW

2.1 Functionality

Like most programs, our system starts in our main function where the input of the file takes place. The input is received through the command line interface and is entered in through the user. With the input, we parse the file to determine either a CPL or IR election and apply the appropriate algorithm to the file by calling the respective class. Once the algorithm has run, the winner of the election is automatically outputted to the screen and the input of the tester's profession (programmer, media member) is used to assess the file that will be produced (audit or media).

List of All Functionalities

- Break Tie
- Export Audit File
- Checking File format
- Identify File
- Read in File
- Prompting User Input
- IR Implementation
- CPL Election Implementation
- Starting the System
- Display Winner
- Audit File
- Processing the ballots file
- Popularity Wins with no Majority in IR
- Write Audit File

2.2 Assumptions

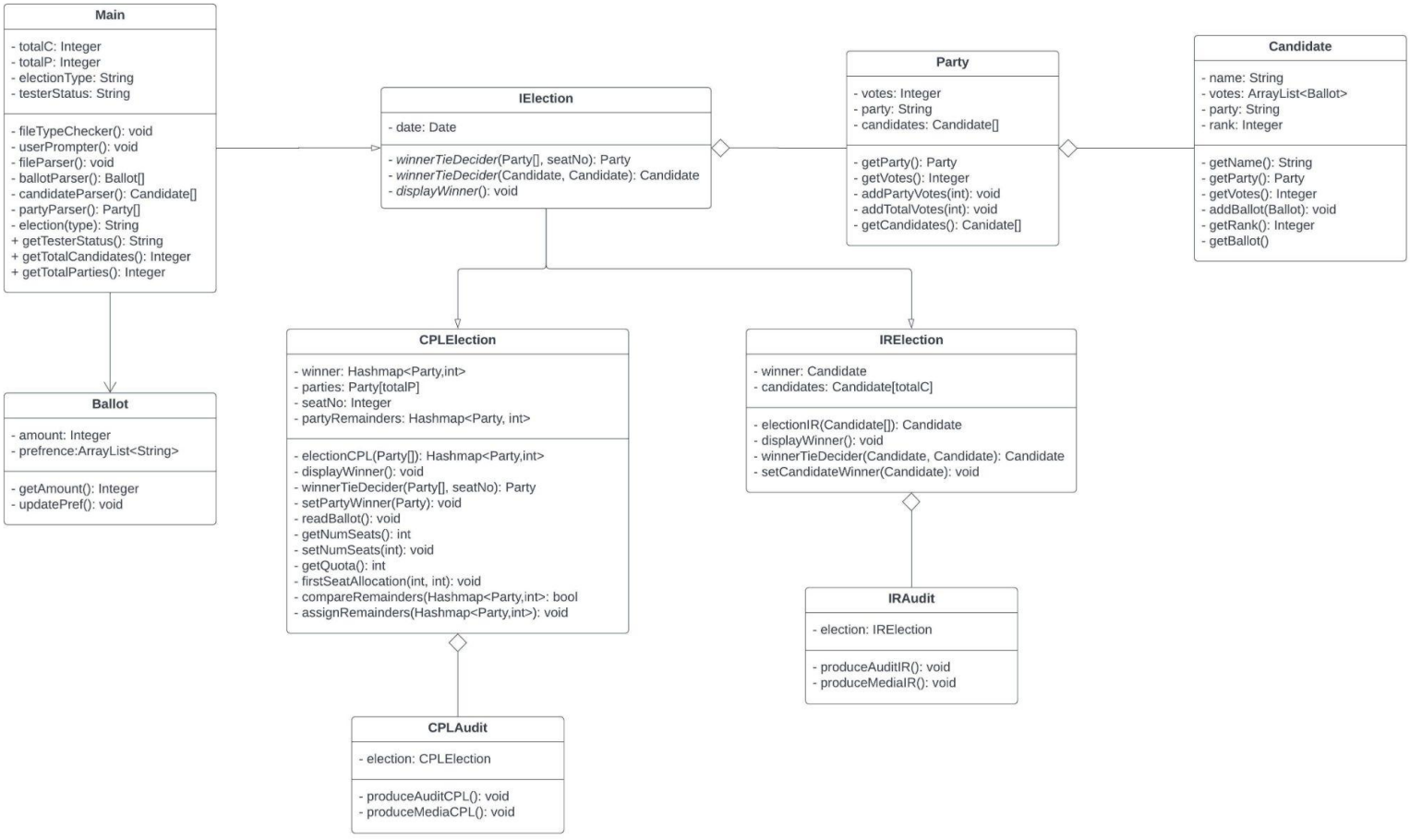
- No more than one file is given per election
- First line of the ballot file will always contain the type of election
- No safety or security requirements are expected
- No errors in the ballots
    - For Instant Runoff ballots specifically, at least one candidate is ranked
    - Ranking Numbers will not have issues (for example, if there are 4 ranks, then expect candidates to be ranked 1,2,3,4 as oppose to a missing 3rd candidate)
    - Voting numbers are 100% accurate

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

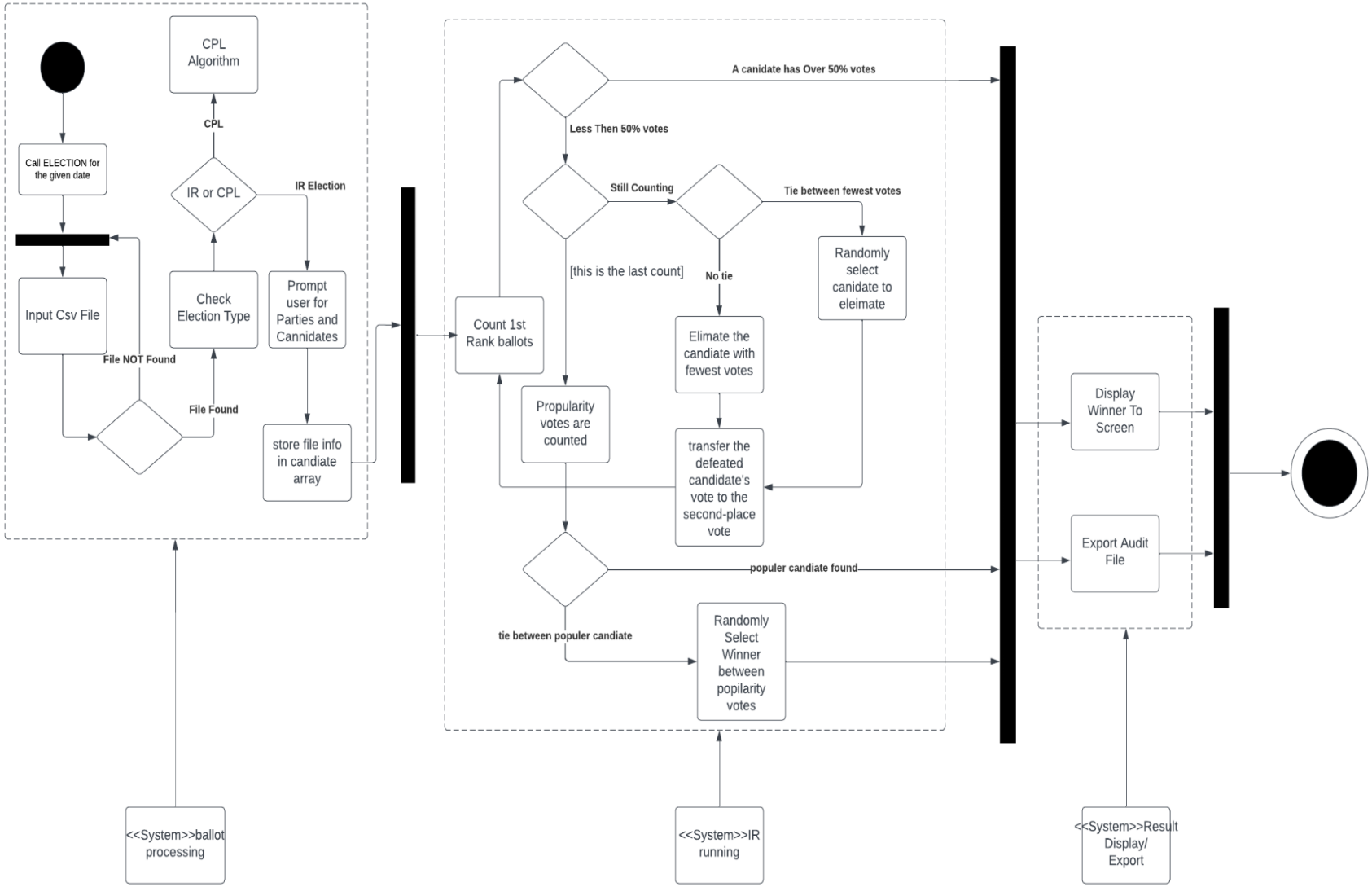
The program is designed to manage the storage and processing of data related to political parties, candidates, and election processes. It consists of several subsystems that work together, including a main class responsible for parsing a CSV file, and two classes, Party and IElective, that inherit from Main. Party is responsible for storing information about a political party and its candidates in an election, while Candidate has an aggregation relationship with Party and stores candidate information. IElective is an abstract class that represents an elective process and runs the election. There are two classes that derive from IElective: IR Election and CPL Election, which implement their respective election algorithms. CPL and IR Election classes have an aggregation relationship with their respective Audit Classes, which are CPLAudit and IRAudit. The system also includes a Ballot class that has an association relationship with the Main class. The overall structure of the system can be visualized as a diagram, with Main as the central node and Party and IElective as its children. IR Election and CPL Election derive from IElective, and CPLAudit and IRAudit are part of their respective classes. The subsystems collaborate with each other through inheritance and aggregation relationships. Party and IElective inherit from Main, while IR Election and CPL Election derive from IElective.

3.2 Decomposition Description



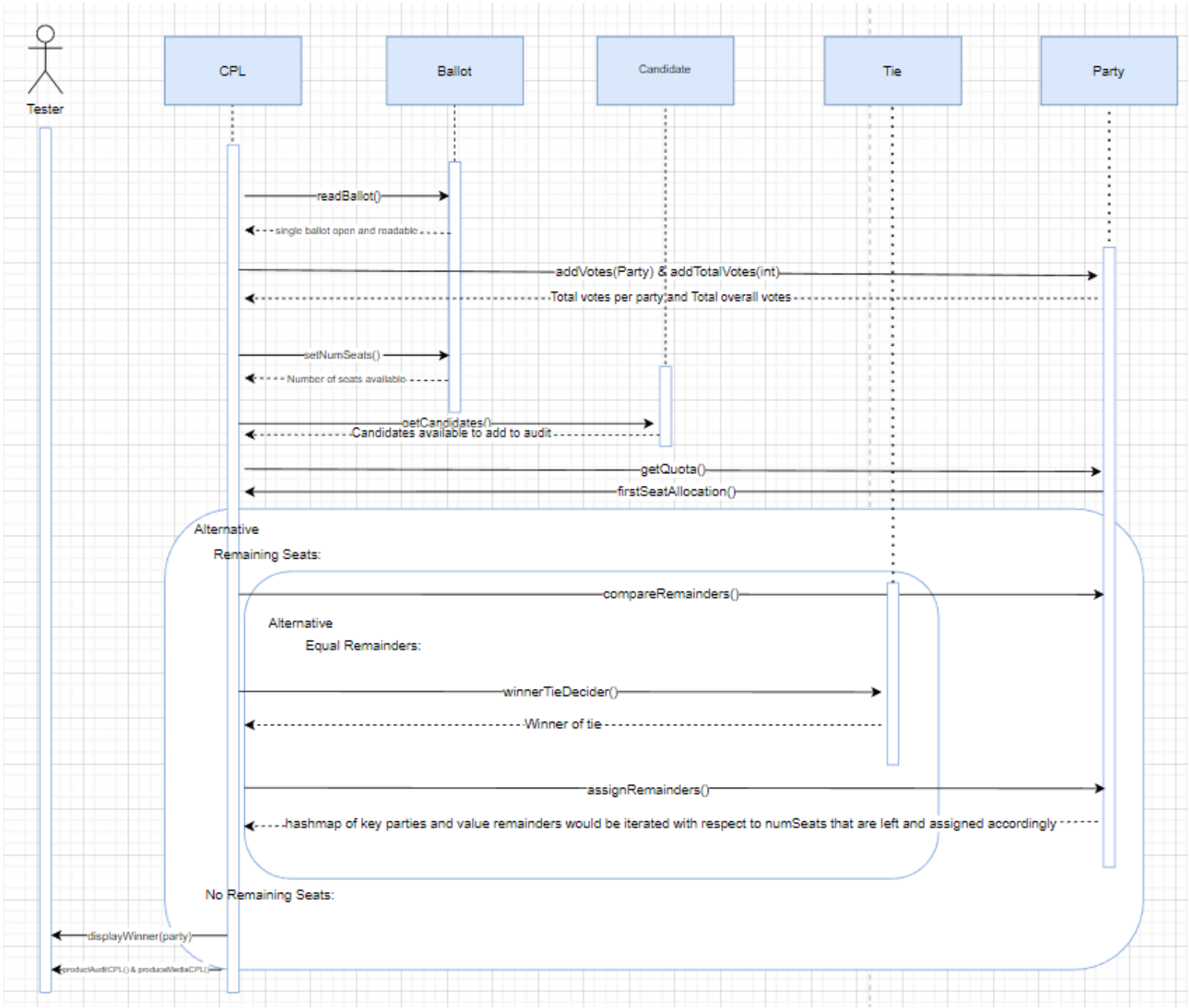
*Note about UML class diagram:

This diagram illustrates the classes required for the system to do its task. We have separated the ballots, candidates, and parties into separate classes based on how they will be used by each election type. The election types have also been represented as separate classes due to the nature of how a winner is determined for each type of election. The elections then have separate audit classes which are used for determining different outputs depending on user input. For more information on the system structure, see section 4.


CPL Algorithm
CPL
Call ELECTION for the given date
IR or CPL
IR Election
Input Csv File
Check Election Type
Prompt user for Parties and Cannidates
File NOT Found
File Found
store file info in candiate array
<<System>>ballot processing
A canidate has Over 50% votes
Less Then 50% votes
Still Counting
Tie between fewest votes
[this is the last count]
No tie
Randomly select canidate to eleimate
Count 1st Rank ballots
Elimate the candiate with fewest votes
Propularity votes are counted
transfer the defeated candidate's vote to the second-place vote
populer candiate found
tie between populer candiate
Randomly Select Winner between popilarity votes
<<System>>IR running
Display Winner To Screen
Export Audit File
<<System>>Result Display/ Export

*Note about UML class diagram:

This activity diagram explains the process of an Instant Runoff election. It begins with the election being called, followed by the input of the CSV file and check the file type. If it's an IR election the voting system program runs the IR algorithm, which counts all the first-place votes on the ballots. If any candidate receives over 50% of the total votes, they are declared the winner. If no candidate has over 50% of the votes, the system checks if it's the last count. If it is, the system counts the popularity votes and declares the popular candidate the winner, unless there is a tie, in which case a random winner is selected. If it's not the last count, the system checks which candidate has the lowest votes to eliminate redistributing their votes to second-place votes, and keeps counting until a candidate receives over 50% of the votes.


Tester
CPL
Ballot
Candidate
Tie
Party
readBallot()
single ballot open and readable
addVotes(Party) & addTotalVotes(int)
Total votes per party and Total overall votes
setNumSeats()
Number of seats available
getCandidates()
Candidates available to add to audit
getQuota()
firstSeatAllocation()
Alternative
Remaining Seats:
compareRemainders()
Alternative
Equal Remainders:
winnerTieDecider()
Winner of tie
assignRemainders()
hashmap of key parties and value remainders would be iterated with respect to numSeats that are left and assigned accordingly
No Remaining Seats:
displayWinner(party)
produceAuditCPL() & produceMediaCPL()

*Note about CPL sequence diagram:

This starts with the assumption that the ballot file has already been inputted and parsed for its corresponding election type (in this case, CPL). This explains why the life of the tester has already begun without instruction.

A more concise pseudocode version of the CPL algorithm can be found in section 5, which highlights the use of each function above.

3.3 Design Rationale

We selected a modular design architecture for our system as it allows for the decomposition of the system into smaller, more manageable subsystems, making the codebase easier to understand, modify, and maintain. This design also enables more efficient testing and debugging since individual components can be tested in isolation.

One critical issue that we considered when selecting this architecture was the need for flexibility and extensibility to handle elections using different algorithms. We needed an architecture that could support the addition of new algorithms in the future without extensive modifications to the existing codebase. A modular design, with a clear separation of concerns between different components, was the best way to achieve this. However, some trade-offs must be considered, such as potential performance overhead when accessing multiple modules.

Regarding the classes, we used a combination of inheritance and aggregation to achieve the desired functionality. The central class responsible for parsing the CSV file is Main, which Party and IElective inherit from. Meanwhile, IR Election and CPL Election derive from IElective. Audit Files are derived from their respective Elections (CPL or IR) and are created in their own classes, CPLAudit and IRAudit.

We also added a Ballot class, which has an association relationship with the Main class. Ballot stores information about an individual's preferences in the election. The addition of this class allows for more complex election algorithms to be implemented, such as preferential voting systems. The use of association relationship allows the Ballot class to be used in the election algorithms without having to modify existing classes.

4. Data Design

4.1 Data Description

The Data we are using is stored in a Comma Separated Value (CSV) file which will be provided by the user and inputted into the system. This file is specifically formatted so that to denote the voting type, candidates and ballots in a way the system can compute.

4.2 Data Dictionary

Find the first draft of items we anticipate to be within our data below:

Data Item	Data Type	Description
Main	Class	The central class of the project where the user input is taken and all classes regarding the election are created
totalC	Integer	Total candidates of election being executed; gain from user input
totalP	Integer	Total parties of election being executed; gain from user input
electionType	String	The type of election to be executed; gained from user input
testerStatus	String	The status of the user which could either be election official or media. Used to determine if media file is generated or not
fileTypeChecker()	Function	Method for validating the format of file being inputted
userPrompter()	Function	Method for prompting the user for inputs including, total number of candidates, parties, election type, tester status and election file.
candidateParser()	Function	Method for parsing and reading inputted file, returning an array of all candidates
partyParser()	Function	Method for parsing and reading inputted file, returning an array of all parties
ballotParser()	Function	Method for parsing and reading inputted

		file, returning an array of the ballots
election(String)	Function	Method for deciding what Voting method algorithm is being done in this election and therefore what election type the system will be handling
Candidate	Class	The Class for the Candidate Object
name	String	The name of the candidate,
votes	Ballot	For the IR election, represents number of votes a candidate received and preference of votes for candidate
party	Party	The party of the candidate, part of the Candidate constructor
rank	Integer	The rank of a candidate for an Closed Party Listing Election
getName() getParty() getVotes() getRank() getBallots()	Function	Getter methods for name, party, total number of votes, rank attributes for CPL, and Ballots for IR
addBallot()	Function	Takes Ballot as parameter, method to add the Ballot of a candidate who was eliminated and their votes are to be transferred according to preference
Party	Class	Class for the Party Object, used to instantiate Republican, Democrat and Independent party objects
votes	Integer	Number of votes allocated to a party object; parameter in Party class constructor
party	String	Name of party; parameter in Party class constructor
candidates	Candidate[]	Array of candidates that are in the respective Party

getParty()	Function	Method to return the name of the Party
getVotes()	Function	Method to return how many votes a party has
addPartyVotes()	Function	Method to add votes to the party object
addTotalVotes()	Function	Method to add votes to the party object after vote retribution from cycle of CPL ELection, accounting total votes
Election	Abstract Class	The abstract class of Election, holds functions for tiebreaking and displaying winner which both Election Types inherit
date	Date	The Date of the election
winnerTieDecider(Party[], seatNo)	Function	Abstract method for deciding tiebreaker for CPL Election
winnerTieDecider(Candidate, Candidate)	Function	Abstract method for deciding tiebreaker for IR Election
displayWinner()	Function	Method to display the winner of the election to the terminal
IRElection	Class	The instant Runoff election class which inherits from Election; its algorithms embodies the instant runoff election requirements
winner	Candidate	Candidate object of the Winner of the election
candidates	Candidate[]	Array of Candidates representing all the candidates in the election, utilizes the totalC object to instantiate the size of the array
electionIR(Candidate[])	Function	Method to actually run the Instant Runoff Election, takes the array of Candidates object *candidate*, and it decides the winner of the election IR style between the candidates,

winnerTieDecider(Candidate, Candidate)	Function	Abstract method inherited from Election class, contains the flipping coin algorithm for deciding tiebreakers between candidates; takes the two candidates with the tie as parameters
setCandidateWinner(Candidate)	Function	Method to set winner of election of the Election object
CPL Election	Class	The closed party listing election class which algorithms embodies the closed party election requirements
winners	HashMap<Party, Integer>	HashMap object of the winners of the CPL Election, with the Party object key, being mapped to the seats integer which represents how many seats the party won
parties	Party[totalP]	Array of all Parties participating in the election
seatNO	Integer	The amount of seats remaining, to be updated during the Election Cycle
partyRemainder	HashMap<Party, Integer>	Remainder of parties that have yet to be given seats, variable to be used and updated through process of CPL Election
readBallot()	Function	Opens ballot up for reading
getNumSeats()	Function	Return the number of seats
setNumSeats(int)	Function	Set the number of seats when seats are being allocated
getQuota()	Function	Returns quota by taking total votes / number of seats
firstSeatAllocation(int, int)	Function	Allocates the first wave of seats by taking the number of votes a certain party has and dividing it by the quota, round this number down to the nearest whole number and that decides how many seats are needed to be distributed for this first round
compareRemainders(H	Function	Iteratively compares if two values

ashmap<Party,int>		(remainders) are equal and determines if a tie needs to be done, returns a bool True if needed to be done also goes to assignRemainders
assignRemainders(Hashmap<Party,int>): void	Function	Assigns the remaining seats now that no ties are made, gives priority to greater remainders and assigns them according to how many seats are left
electionCPL(Party[])	Function	Method which takes in all the parties participating in the election of the parameter and ascertains who wins what seats under the Close Party Listing voting method. Returns HashMap<Party, Integer> winner variable
displayWinner(HashMap<Party, Integer>)	Function	Takes in Hashmap of Party matched with how many seats won and determines the candidate winners of the parties based on the seats of the parties and the candidate ranks
winnerTieDecider(Party, Party)	Function	Tiebreaker method which utilizes a flip-a-coin algorithm when there is a tie between parties in CPL
setPartyWinner(HashMap<Party, Integer>)	Function	Sets the winner of the CPL Election object including a HashMap making note of all parties and how many seats they won
CPL Audit	Class	Class for producing the audit for the CPL election
produceAuditCPL()	Function	Method for producing and writing the actual Audit File of the processed CPL Election
IR Audit	Class	Class for producing the audit for the IR election
produceAuditIR()	Function	Method for producing and writing the actual Audit File of the processed IR Election

Ballot	Class	Class for holding the ballots containing votes for candidates along with the ranked preference for candidates among those votes
amount	Integer	The amount of votes
preference	ArrayList<String>	The ordered list of preferred candidates the votes would default to if their number one option were eliminated
getAmount	Function	Method to return number of votes
updatesPref()	Function	Method to give votes to the second preferred candidate of the array list if the firstmost was eliminated
CPLAudit	Class	Audit class for CPL Election used to write media and audit files
election	CPLElection	CPLElection object to be used to produce Audit and Media File
IRAudit	Class	Audit class for IR Election used to write media and audit files
election	IRElection	IRElection object to be used to produce Audit and Media File

5. COMPONENT DESIGN

PseudoCode

Prompt for userInput(numberOfCandidates, numberOfParties, electionType , electionFile)

If checkFileType(Election File) = **True** Then **Proceed**

Else Prompt for userInput(Again)

If electionType = **IR**

Then execute **Case 1**

Else execute **Case 2**

In Case 1 create IR_Election(candidates[numberOfCandidates], winner)

Run IR_Election algorithm

For each candidate in the election

If this.candidate.ballot.votes > every.other.candidate.ballot.votes.**combined**

Then this.candidate.ballot.votes.majority() = **True**

Set this.candidate = winner **And Return** this.candidate

Else removeFromElection(that.candidate) **Where**

that.candidate.ballot.votes < every.other.candidate.ballot.votes

For each candidate in the election

If Tie Then flipCoin(candidate1, candidate2) **And Return** that.candidate

END FOR

AND that**.**candidate.ballot.nextPreference() **Inherits** that.candidate.ballot

Else If candidates.remaining() = 2 **AND** this.candidate.ballot.votes.majority() = False **Then** flipCoin(Candidate 1, Candidate 2) **And Return** this.candidate

Set candidate **As** winner

End For

Run displayWinner(winner)

Run produceAudit(IR_Election)

End Case 1

In Case 2 create CPL_Election(parties[numberofParties, winner)

Run CPL_Election Algorithm

Read ballot (readBallot())

For each party

Add votes per party (addVotes(Party))

Add votes to total (addTotalVotes(int))

Set the number of seats from ballot file (setNumSeats())

Calculate quota by dividing total votes by number of seats (getQuota())

Allocate first wave of seats by taking quota // votes per party and subtract number of Seats by those that are allocated(firstSeatAllocation())

If there are remaining seats (compareRemainders())

If remainders are equal

Then flip coin to break tie (winnerTieDecider())

Else

Assign the remaining seats from greatest remainder to least

Runs displayWinner(winner)

Runs produceAudit(CPL_Election)

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user will be able to interact with the system by providing any additional information needed for the election results to be calculated. This provided information will be used to run the system in a more efficient manner. The information provided allows for arrays to be created of the correct size before any of the parsing is done in the file.

6.2 Screen Images

Example of terminal:

```
-Program Running-

What is the Election Type
-
How many Parties are there
-
How many Candidates are there
-
What is your role (i.e Tester or Media)
```

```
-Program Complete-
-Results Posted in Terminal-

The winner of the election is: "..."
or
The winning party of the election is: "..."

```

6.3 Screen Objects and Actions

The way in which the user may interact with the system is by simply typing information based on what they are prompted. The system will then use this information to better determine how it will approach calculating a winner based on the algorithm. For example, the user can specify whether they are a media personnel or simply a tester. If they are a part of the media, an Audit file will not be produced, as it is not necessary for what they need from the system. On the other hand, if the user specifies that they are a tester, an Audit file will be produced.

7. REQUIREMENTS MATRIX

ID	Use Case	Requirement
GT_001	Break Tie	• Instantiated in the *Election* class in the method of *winnerTieDecider()* • Also found in the *CPL Election* & *IR Election* classes which inherit the method
GT_002	Export Audit File	• Instantiated in the *CPL Audit* & *IR Audit* classes
GT_003	Checking File format	• Implemented in the *Main* class *filetypechecker* method
GT_004	Identify File	• Implemented in the *fileParser()* & *fileTypeChecker()* method of the *Main* class
GT_005	Read in File	• Implemented in the *Main* class's *candidateParser()* & *partyParser()* methods
GT_006	Prompting User Input	• Implemented in the *userPrompter()* of the *Main* method
GT_007	IR Implementation	• Implemented in the *IR Election* class and starts from the *String type* object in the *Main* class
GT_008	CPL Election Implementation	• Implemented in the *CPL Election* class and starts from the *String type* object in the *Main* class
GT_009	Starting the System	• Initiated in the *Main* class and more specifically *userPrompter()* method which is the first ran function of the program
GT_010	Display Winner	• Instantiated in the *Election* class in the method of *displayWinner()* • Also found in the *CPL Election* & *IR Election* classes which inherit the method
GT_011	Audit File	• Implemented in the *CPL Audit* & *IR Audit* classes
GT_012	Processing the ballots file	• Implemented in the *Main* class's *fileParser()* method

GT_013	**Popularity Wins with no Majority in IR**	● **This is accounted for in the *electionIR* method in the *IR Election* class**
GT_014	**Write Audit File**	● **Implemented in the *produceAuditCPL()* & *produceAuditIR()* methods in the *CPL Audit* & *IR Audit* classes**