Team#21

Waterfall Evaluation System

Software Design Document

Name (s): Andrew Guerra, Evan Bagwell, Hady Kotifani, Andrew Guerra, Tyler Grimm

Date: 03/03/2023

Table of Contents

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of the Waterfall Election Evaluation system. It will explain the purpose and feature of the design of the system and show how the system functions and how all classes interact with each other.

The expected audience for this document are the users of the system, programmers who are working on the system, and testers. Each will gain an understanding of the system by reading this document, and testers will better understand how to write tests by reading this document and understanding how the whole system functions.

1.2 Scope

This document contains an in-depth description of the system design of the Waterfall Election Evaluation system. The Waterfall Election Evaluation system is designed to determine election results for either Instant Runoff (IR) or Closed Party List (CPL) elections. It is expected to run multiple times during the year at normal elections time and during special elections. This system will be designed to be fair in its election determinations. The benefits of the Waterfall Election Evaluation are that it allows multiple election types to be processed through a single program and also removes manual processes by automating the counting of votes and determining the winners.

Additionally, the system will produce an audit file containing information about winners and the processing of ballots so that media and auditors may review how the election unfolded. Its purpose is to instill trust in the system through transparency and legitimize the results of the election.

1.3 Overview

Section 1 is an Introduction. It contains a brief introduction to the project and a description of this document. The purpose of this document and the scope of the project are discussed. Also contained are references used to write this document, and an overview of the definitions and acronyms used frequently in this document.

Section 2 describes the System Overview. It contains an overview of the functionality, context, and design of the system. Background information is also provided.

Section 3 is the System Architecture. It contains all system diagrams. Those diagrams include UML activity diagrams, a UML class diagram, and a sequence diagram. Also included is the rationale for why we choose to design our system in the way we did.

Section 4 is about Data Design. It contains a description of how the information domain of the system is transformed into data structures, as well as describing how the major data or system entities are stored, processed and organized.

Section 5 contains the Component Design that summarizes each object member function in a more systematic way.

Section 6 is about User Interface Design. It describes the functionality of the system from the user's perspective. Also contained are screenshots showing the interface from the user's perspective, as well as a sample audit file.

Section 7 is the Requirements Matrix. Components and data structure are cross referenced to help show which system components satisfy each of the functional requirements from the SRS.

Section 8 is the Appendix.

1.4 Reference Material

IEEE Recommended Practice for Software Design Descriptions

IEEE Std 10161998

Waterfall Election Evaluation - Software Requirements Specification

Waterfall Election Evaluation - SRS

1.5 Definitions and Acronyms

Term	**Definition**
.csv file	a .csv file is a file with comma-separated values
Audit file	An Audit file will be the file that is produced at the end of an election.
CPL	Stands for Closed Party List Voting, CPL process described in Appendix A
Coin Flip	A coin flip is a random pick chosen fairly that will be used to determine the winner of an election given that there is a tie.
IRV or IR	Stands for Instant Runoff Voting, IR process described in Appendix B

Term	**Definition**
Programmer	A programmer refers to the person(s) responsible for developing the system
System	The term system will refer to the Waterfall Election Evaluation software that this document describes.
Tester	A tester will be a person(s) that is responsible for testing the system and ensuring that it is working properly.

2. System Overview

The system will be able to do the following:

- Accept a file: users will input a file containing all information needed for the election, and the system will check for correct inputs.
- Read the file: the system will read the file and store the information contained in the file.
- Determine Election Type: the system will determine the type of election based on the contents of the previously read file
- Determine Candidates: the system will determine the candidates needed for an IR election or a CPL election
- Determine Ballots: the system will determine the number of ballots in an election
- Determine Seats: the system will determine the number of seats in an election
- Run Election: the system will run an election for either IR or CPL type election
- Run Popularity: the system will run a popularity election if there is no clear majority in an IR election.
- Fair Coin Toss: the system will flip a coin to determine a winner if there is a tie.
- Display Results: the system will display a brief description of the election after a winner is determined
- Generate Audit File: the system will generate an in-depth report about the election.

Background information for the system:

- The system will run on a Linux operating system, specifically version 20.03 of Ubuntu, and is not expected to behave or run properly under other operating systems.
- The system will process CSV files of a specified format and is not expected to be able to process files in a different format.

3. System Architecture

3.1 Architectural Design

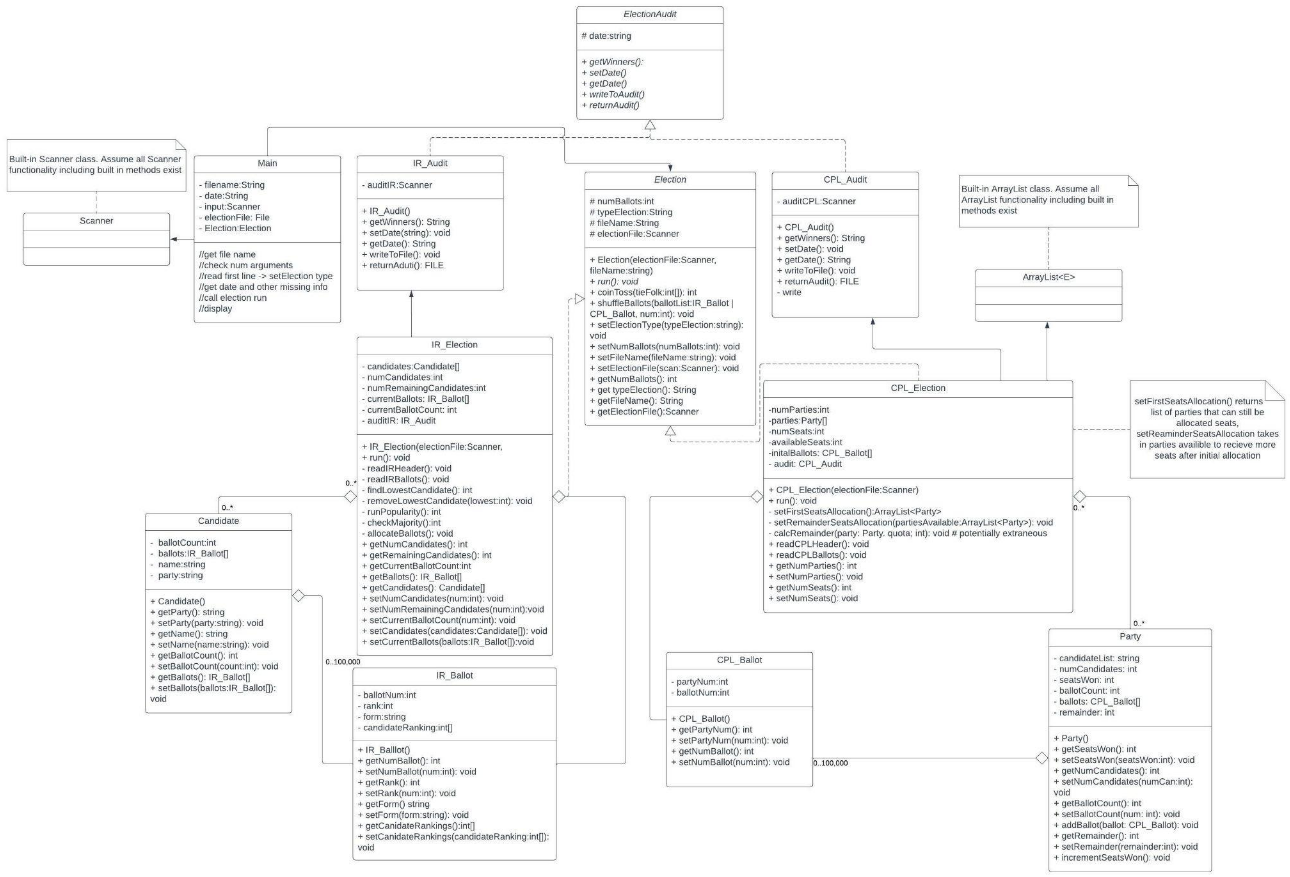

ElectionAudit
# date:string
+ getWinners():
+ setDate()
+ getDate()
+ writeToAudit()
+ returnAudit()
Built-in Scanner class. Assume all Scanner functionality including built in methods exist
Scanner
Main
- filename:String
- date:String
- input:Scanner
- electionFile: File
- Election:Election
//get file name
//check num arguments
//read first line -> setElection type
//get date and other missing info
//call election run
//display
IR_Audit
- auditIR:Scanner
+ IR_Audit()
+ getWinners(): String
+ setDate(string): void
+ getDate(): String
+ writeToFile(): void
+ returnAduti(): FILE
Election
# numBallots:int
# typeElection:String
# fileName:String
# electionFile:Scanner
+ Election(electionFile:Scanner, fileName:string)
+ run(): void
+ coinToss(tieFolk:int[]): int
+ shuffleBallots(ballotList:IR_Ballot | CPL_Ballot, num:int): void
+ setElectionType(typeElection:string): void
+ setNumBallots(numBallots:int): void
+ setFileName(fileName:string): void
+ setElectionFile(scan:Scanner): void
+ getNumBallots(): int
+ get typeElection(): String
+ getFileName(): String
+ getElectionFile():Scanner
CPL_Audit
- auditCPL:Scanner
+ CPL_Audit()
+ getWinners(): String
+ setDate(): void
+ getDate(): String
+ writeToFile(): void
+ returnAudit(): FILE
- write
Built-in ArrayList class. Assume all ArrayList functionality including built in methods exist
ArrayList<E>
IR_Election
- candidates:Candidate[]
- numCandidates:int
- numRemainingCandidates:int
- currentBallots: IR_Ballot[]
- currentBallotCount: int
- auditIR: IR_Audit
+ IR_Election(electionFile:Scanner,
+ run(): void
- readIRHeader(): void
- readIRBallots(): void
- findLowestCandidate(): int
- removeLowestCandidate(lowest:int): void
- runPopularity(): int
- checkMajority():int
- allocateBallots(): void
+ getNumCandidates(): int
+ getRemainingCandidates(): int
+ getCurrentBallotCount:int
+ getBallots(): IR_Ballot[]
+ getCandidates(): Candidate[]
+ setNumCandidates(num:int): void
+ setNumRemainingCandidates(num:int):void
+ setCurrentBallotCount(num:int): void
+ setCandidates(candidates:Candidate[]): void
+ setCurrentBallots(ballots:IR_Ballot[]):void
CPL_Election
-numParties:int
-parties:Party[]
-numSeats:int
-availableSeats:int
-initalBallots: CPL_Ballot[]
- audit: CPL_Audit
+ CPL_Election(electionFile:Scanner)
+ run(): void
- setFirstSeatsAllocation():ArrayList<Party>
- setRemainderSeatsAllocation(partiesAvailable:ArrayList<Party>): void
- calcRemainder(party: Party. quota; int): void # potentially extraneous
+ readCPLHeader(): void
+ readCPLBallots(): void
+ getNumParties(): int
+ setNumParties(): void
+ getNumSeats(): int
+ setNumSeats(): void
setFirstSeatsAllocation() returns list of parties that can still be allocated seats, setReaminderSeatsAllocation takes in parties availible to recieve more seats after initial allocation
0..*
0..*
Candidate
- ballotCount:int
- ballots:IR_Ballot[]
- name:string
- party:string
+ Candidate()
+ getParty(): string
+ setParty(party:string): void
+ getName(): string
+ setName(name:string): void
+ getBallotCount(): int
+ setBallotCount(count:int): void
+ getBallots(): IR_Ballot[]
+ setBallots(ballots:IR_Ballot[]): void
0..100,000
IR_Ballot
- ballotNum:int
- rank:int
- form:string
- candidateRanking:int[]
+ IR_Balllot()
+ getNumBallot(): int
+ setNumBallot(num:int): void
+ getRank(): int
+ setRank(num:int): void
+ getForm() string
+ setForm(form:string): void
+ getCanidateRankings():int[]
+ setCanidateRankings(candidateRanking:int[]): void
0..*
0..*
CPL_Ballot
- partyNum:int
- ballotNum:int
+ CPL_Ballot()
+ getPartyNum(): int
+ setPartyNum(num:int): void
+ getNumBallot(): int
+ setNumBallot(num:int): void
0..100,000
Party
- candidateList: string
- numCandidates: int
- seatsWon: int
- ballotCount: int
- ballots: CPL_Ballot[]
- remainder: int
+ Party()
+ getSeatsWon(): int
+ setSeatsWon(seatsWon:int): void
+ getNumCandidates(): int
+ setNumCandidates(numCan:int): void
+ getBallotCount(): int
+ setBallotCount(num: int): void
+ addBallot(ballot: CPL_Ballot): void
+ getRemainder(): int
+ setRemainder(remainder:int): void
+ incrementSeatsWon(): void

Figure *3.1* Architecture Design

Figure 3.1 describes the architecture design of the election system. The Main class acts as the driver for the program, where two election implementations, CPL and IR, are represented in the classes CPL_Election and IR_Election respectively and compose of the main functionality of the election system. These two classes have aggregate relationships with a type of ballot, and either a candidate or party. The last component of the system is the Audit class, which handles the election data that is passed in as an IR or CPL election is being run. Once an election is complete, an Audit file is created and election results are determined.

3.2 Decomposition Description

3.2.1 IR Activity Diagram

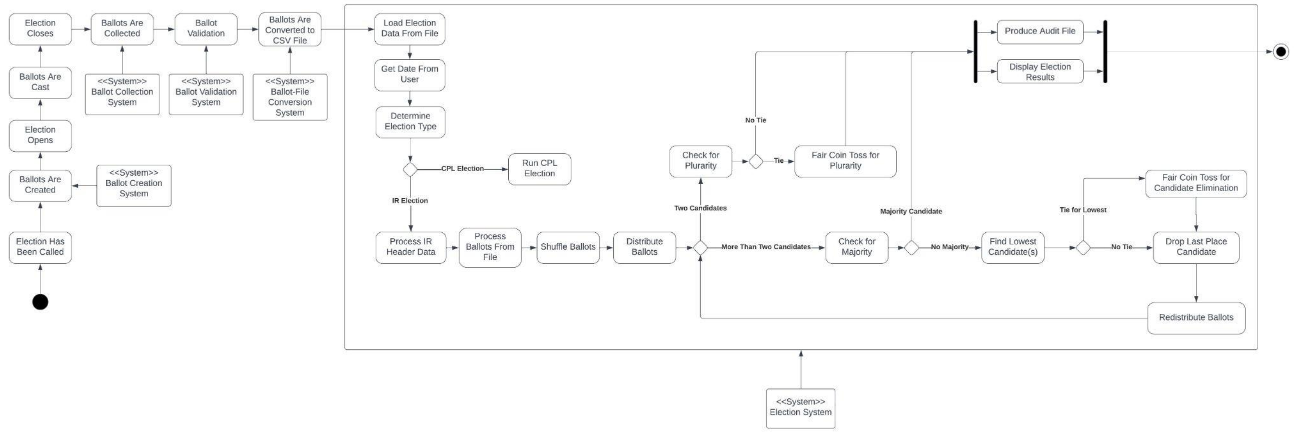


Figure *3.2* IR Activity Diagram

The IR activity diagram begins when an election has been called for a certain date and ends with a winner being determined. The sequence of events are described below:

1. Election Has Been Called → An election has been determined by an election official for a certain date
2. Ballots Are Created → The ballots used for the election are created from an external Ballot Collection System
3. Election Opens → The election is now determined to be open by an election official and ballots are ready to be cast
4. Ballots are Cast → Ballots in the election are cast in the election
5. Election Closes → The election is now determine to be closed by an election official and ballots are no longer casted
6. Ballots Are Collected → Ballots cast in the election are collected through an external Ballot Collection System
7. Ballot Validation → Ballots are run through an external Validation System where ballots with errors are thrown out and not considered in the election
8. Ballots Are Converted to CSV FIle → Ballots are grouped and converted to a CSV file to read through an external Ballot File Conversion System

The following steps follow an IR election within the election system:

1. Load Election Data From File → CSV file containing the ballots is imported into the election system
2. Get Date From User → The date is determined from a prompt to the user
3. Determine Election Type → Election type is determined to be IR from the choice of IR or CPL

4. Process IR Header Data → Relevant information pertaining to the IR election is read from the header of the CSV file
5. Process Ballots From File → Ballots are read from the CSV file and are stored in the system
6. Shuffle Ballots → Ballots are shuffled in the system to ensure randomness
7. Distribute Ballots → Ballots are read and distributed to candidates
8. Check For Plurality → When there are two candidates check whether a plurality from one candidate has been achieved
    a. If one candidate is determined to be the winner, an audit file of the election is produced and election results are displayed to the terminal
    b. If no candidate is determined to be the winner, a fair coin toss for plurality is conducted and the winner of the coin toss is determined to be the winner. An audit file of the election is produced and election results are displayed to the terminal
9. Check For Majority → When there are more than two candidates, check whether a candidate has one an outright majority
    a. If one candidate has an outright majority, the candidate is determined to be the winner. An audit file of the election is produced and election results are displayed to the terminal
10. Find Lowest Candidate(s) → The candidates with the lowest amount of votes from the latest round of the election is determined
    a. If only one candidate has the lowest number of votes, the candidate is dropped from the election
    b. If more than one candidate has the lowest number of votes, a fair coin toss determines that candidate to be eliminated. The eliminated candidate is dropped from the election
11. Redistribute Ballots → Ballots from dropped candidates are redistributed to their next ranked candidate. The election process for IR then continues from step. 7 of the election system

3.2.2 CPL Sequence Diagram

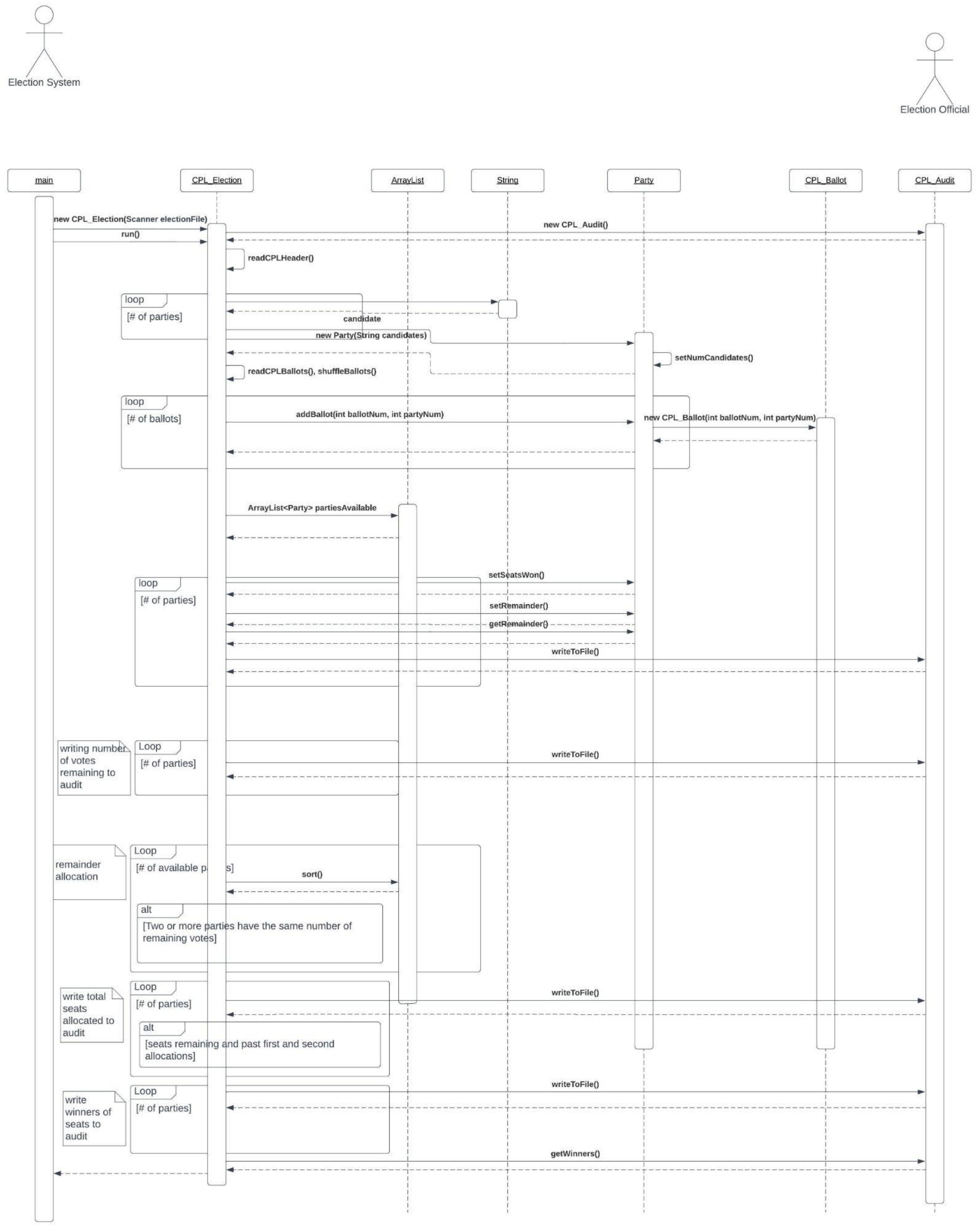

Election System
Election Official
main
CPL_Election
ArrayList
String
Party
CPL_Ballot
CPL_Audit
new CPL_Election(Scanner electionFile)
run()
new CPL_Audit()
readCPLHeader()
loop
[# of parties]
candidate
new Party(String candidates)
setNumCandidates()
readCPLBallots(), shuffleBallots()
loop
[# of ballots]
addBallot(int ballotNum, int partyNum)
new CPL_Ballot(int ballotNum, int partyNum)
ArrayList<Party> partiesAvailable
loop
[# of parties]
setSeatsWon()
setRemainder()
getRemainder()
writeToFile()
writing number of votes remaining to audit
Loop
[# of parties]
writeToFile()
remainder allocation
Loop
[# of available parties]
sort()
alt
[Two or more parties have the same number of remaining votes]
write total seats allocated to audit
Loop
[# of parties]
alt
[seats remaining and past first and second allocations]
writeToFile()
write winners of seats to audit
Loop
[# of parties]
writeToFile()
getWinners()

Figure *3.3* CPL Sequence Diagram

The CPL sequence diagram begins the moment of receiving the ballots into the system from the file, and ends when a winner has been declared. The diagram first describes the initialization of a CPL_Election object that is passed a Scanner or read in the ballots of the CSV file. The constructor also initializes an audit file that will be called using the write() function to add data to the audit file as the election progresses.

After the constructor, in main, the run() method in CPL_Election runs and acts as a driver for the rest of the CPL election. First, a helper is called to read in information regarding the election from the header of the CSV file and the relevant fields within CPL_Election are populated as a result. The parties for the election are then created and stored in an array with proper ordering.

The next step in the sequence is to read in the ballots from the CSV file to be stored in an Array and the elements in the structure are then sorted. Next, we initialize a temporary ArrayList to hold parties still left in contention, and then for each party the number of seats won is calculated from the quota along with the remainder of seats. The remainder seats are then allocated to their respective parties, and CPL_Audit is passed information regarding total seats allocated to each party, along with the winners of the seats (the candidates of each party). getWinners() is called on the CPL_Audit object to display the election results.

3.3 Design Rationale

Ballot Collection Rationale: One decision that was made was with regards to the process of reading the ballots into the system. It was determined that reading in the ballots within the framework of the respective election class vs reading in the ballots in an additional class before being interpreted by an election class. Reading in the ballots within the framework of an election class allows us to avoid situations where we have to interpret how a ballot should be read as it may not be initially clear what election is present as the ballots are different depending on the election type. Determining the election and having the read in of the ballots within its respective election class avoids any ambiguity as to how to handle ballots.

Audit Creation Rationale: One design issue was whether to implement a data structure to represent the contents needed to write the audit file before having the audit class take in such data structure and create the audit file from it. However, it is much easier instead of saving a data structure for the audit file contents, we write directly to the audit file as an election progresses via a function call to the audit class. This avoids using additional memory to hold the contents of the audit file before writing to the file. The one drawback of this solution is that it involves many calls to writing to a file where with larger data sets this has the potential to be slow.

4. Data Design

4.1 Data Description

The ballots are read in from the CSV file and are stored in an array by parsing the file and storing. The system does not interact with any databases. The CSV file will be located in the current working directory.

4.2 Data Dictionary

4.2.1 Candidate Class

Candidate is a class designed to store the information about candidates for an IR election.

Candidate contains the following attributes:

- ballotCount: a private integer that stores the number of ballots the candidate has received
- ballots: a private array of IR_Ballot
- name: a private string that contains the candidates name
- party: a private string that contains the candidates party

and the following methods:

- Candidate(): a public constructor of Candidate class.
- getParty(): a public method that returns the candidate party, has return type string
- setParty(): a public method that sets the candidates party, return type void
- getName(): a public method that returns the candidates name, has return type string
- setName(): a public method that sets the candidates name, return type void
- getBallotCount(): a public method that returns the candidate ballot count, has return type integer
- setBallotCount(): a public method that sets the candidates ballot count, return type void
- getBallots(): a public method that gets the list of Ballot object a candidate has
- setBallots(): a method that allows the iser to pass a candidate a list of Ballot object that represent what ballots a user gets

4.2.2 Parties Class

Parties is a class that contains information needed about parties in a CPL election.

Parties contains the following attributes:

- candidateList: a private string containing the list of candidates belonging to a party
- numCandidates: a private int that stores the number of candidates belonging to a party
- seatsWon: a private int that stores the number of seats a party has won in a CPL election
- ballotCount: a private int containing the number of ballots
- ballots: a private array of CPL_Ballot

- Remainder: a private attribute that holds the number of remaining votes after calculating the quota

and the following methods:

- Parties(): a public constructor of Parties class
- getSeatsWon(): a public method that returns the number of seats won, return type integer
- setSeatsWon(): a public method that sets the number of seats won, return type void
- getNumCandidates(): a public method that returns the number of candidates in a party, return type integer
- setNumCandidates(): a public method that sets the number of candidates in a party, return type void
- getBallotCount(): a public method that gets the ballot count, return type integer
- setBallotCount(): a public method that sets the ballot count, return type void
- addBallot(): adds the ballot to the party
- getRemainder(): returns the remainder attribute
- setRemainder(): a public method that sets the remainder attribute of the Party class to some integer given
- incrementSeatsWon(): increments the number of seats that the Party has won

4.2.3 IR_Ballot Class

IR_Ballot is a class that contains the information needed for ballots in an IR Election

IR_Ballot contains the following attributes:

- ballotNum: a private integer containing the total number of ballots
- rank: a private integer containing the rank
- Form: holds the rankings of candidates submitted by a voter as a single string in the form of #,#,#,#,# such as ,,,1,2 or 1,2,,3,, etc.
- candidateRanking: private array of string the contains all candidates rank

and the following methods:

- IR_Ballot(): a public construct of IR_Ballot class
- getNumBallot(): a public method to get the number of ballots, return type integer
- setNumBallot(): a public method to set the number of ballots, return type void
- getRank(): a public method to get the rank, return type integer
- setRank(): a public method to set the rank, return type void

4.2.4 CPL_Ballot Class

CPL_Ballot is a class that contains the information needed for ballots in a CPL Election

CPL_Ballots contains the following attributes:

- partyNum: a private integer that holds the number of parties

- ballotNum: a private integer that holds the number of ballots

and the following methods:

- CPL_Ballot(): a public constructor of CPL_Ballot class
- getPartyNum(): a public method to get the number of parties, return type integer
- setPartyNum(): a public method to set the number of parties, return type void
- getNumBallot(): a public method to get the number of ballots, return type integer
- setNumBallot(): a public method to set the number of ballots, return type void

4.2.5 Election Class

Election is a class that contains the information needed to execute an election

Election contains the following attributes:

- numBallots: a protected integer the contains the number of ballots
- typeElection: a protected string that has the election type
- fileName: a protected string that contains the CSV file name
- electionFile: a protected Scanner to read in the file

and the following methods:

- Election(electionFile:Scanner): public Election Constructor, takes in electionFile as a parameter
- run(): abstract method to run the election, return type void
- coinToss(tieFolk:int[]): public method to execute a coin toss, takes in the array of people who have tied
- shuffleBallots(ballotList:B): public method to shuffle the ballots, takes in an array of ballots, return type void
- setElectionType(typeElection:string): public method to set the election type, return type void
- setNumBallots(numBallots:int): public method to set the number of ballots, return type void
- setElectionFile(scan:Scanner): public method to se the electionFile that is used to read in the file

4.2.6 IR_Election Class (inherits Election)

IR_Election is a class that contains information needed to run an IR_Election

IR_Election contains the following attributes:

- Candidates: a private array of Candidate, holds the candidate for an election
- numCandidates: a private integer that holds the number of candidates in an election
- currentBallots: a private array of IR_Ballot, hold IR ballots that need to be distributed

- currentBallotCount: the number of ballots to be distributed to candidates
- auditIR: a private attribute of array type and holds information and control for printing information to the audit file

and the following methods:

- IR_Election(): a public constructor of IR_Election class
- run(): a public method that runs the election, return type void
- readIRHeader(): a private method that reads the header of the IR ballot file, return type void
- readIRBallots(): a private method that reads the IR ballot and modifies attributes, return type void
- findLowestCandidate(): a private methods that finds the candidate with the fewest number of votes, returns void
- removeLowestCandidate(lowest:int): a private method that removes the candidate with the lowest number of votes from the running and removes the ballots attributed to that candidate
- checkMajority(): a private method that checks if any of the candidates has a majority in votes and returns the candidate number that won or another value if there is no clear majority, returns an int
- allocateBallots(): a private method that gives the candidates their associated ballots, returns void
- runPopularity(): a public method to run the popularity algorithm, return type void
- getNumCandidates(): a public method that gets the number of candidates, return type integer
- getRemainingRemainingCandidates: a public method that gets the number of candidates, returns an integer
- getCurrentBallotCount(): a public method that gets the number of ballots that need to be allocated
- getBallots(): a public method that return the list of IR_Ballots that have yet to be allocated, type IR_Ballot[]
- getCandidates(): a public method that returns the list of candidates that are still in the running
- setNumCandidates(num:int): a public method that sets the number of candidates, return type void
- setNumRemainingCandidates(num:int): a public method that sets the number of candidates that are still in the running, type void
- setCurrentBallotCount(num:int): a public method that sets the current ballot count to the wanted number, returns void
- setCandidates(candidates:Candidate[]): a public method that sets the candidates attribute to a new list of candidates, returns void
- setCurrentBallots(ballots:IR_Ballot[]): a public method that sets the currentBallots attribute to a new list of ballots, returns void

4.2.7 CPL_Election Class (inherits Election)

CPL_Election is a class that contains information needed to run a CPL Election

CPL_Election contains the following attributes:

- numParties: is a private integer that contains the number of parties
- parties: is a private array of Party, used to store all the parties
- numSeats: is a private integer that contains the number of seats
- availableSeats: is a private integer that contains the number of available seats
- initalBallots: is a private array of CPL_Ballot, used to store the ballots
- audit: CPL_Audit, used for generating audit file

and the following methods:

- CPL_Election(electionFile:Scanner): public CPL_Election constructor, takes in electionFile as parameter
- run(): public method to run the election, return type void
- setFirstSeatsAllocation(): private method to set the first seats, returns ArrayList<Party>
- setRemainderSeatsAllocation(partiesAvailable:ArrayList<Party>): private method the set the remainder of seat allocation, takes partiesAvailable as a parameter, return type void
- calcRemainder(party: Party. quota; int): private method to find the remainder, takes party and quota as parameters, return type void
- readCPLHeader(): public method that reads the CPL header, return type void
- readCPLBallots(): public method that reads the CPL ballots, return type void
- getNumParties(): public method to get the number of parties, return type integer
- setNumParties(): public method to set the number of parties, return type void
- getNumSeats(): public method to get the number of seats, return type integer
- setNumSeats(): public method to set the number of seats, return type void

4.2.8 ElectionAudit Class

ElectionAudit contains what is needed to produce an election audit file

ElectionAudit has the following attributes:

- date: protected string that has the date

and the following methods:

- getWinners(): abstract method to get the winners
- setDate(): abstract method to set the date
- getDate(): abstract method to get the date
- writeToAudit(): abstract method to write to the audit file
- returnAudit(): abstract method to return the audit file

4.2.9 IR_Audit Class (inherits ElectionAudit)

IR_Audit contains what is needed to produce an audit file for IR election

IR_Audit contains the following attributes:

- auditIR: private Scanner to read in auditIR

and the following methods:

- IR_Audit(): public IR_Audit constructor
- getWinners(): public method to get the winners, return type String
- setDate(string): public method to set the date, takes in string as a parameter, return type void
- getDate(): public method to get the date, return type string
- writeToFile(): public method to write to the audit file, return type void
- returnAduti(): public method to return the audit file, return type FILE

4.2.10 CPL_Audit Class (inherits ElectionAudit)

CPL_Audit contains what is needed to produce an audit file for CPL election

CPL_Audit has the following attributes:

- auditCPL: private Scanner to read in auditCPL

and the following methods:

- CPL_Audit(): public CPL_Audit constructor
- getWinners(): public method to get the winners, return type String
- setDate(string): public method to set the date, takes in string as a parameter, return type void
- getDate(): public method to get the date, return type string
- writeToFile(): public method to write to the audit file, return type void
- returnAduti(): public method to return the audit file, return type FILE

5. Component Design

The designs and pseudocode for each member function of the classes involved in our system are detailed in the remainder of this section. The member functions are outlined for each of the classes, main, Election, IR_Election, IR_Ballot, Candidate, CPL_Election, CPL_Ballot, Party, CPL_Audit, and ElectionAudit. Section 3.2 shows the flow of data and interactions with these objects and methods.

Main

int main (int args, string args[])

```
    String fileName

    String date

    Scanner input = new Scanner
```

```
if (args < 2)
    Print ("Enter the file name")
    fileName = input from keyboard
else
    fileName = args[1]
Print("Enter date of election in format mm/dd/yyyy")
date = input from keyboard
File electionFile = open(fileName)
If (electionFile == Null)
    Print("file failed to open")
    Return 1 //error opening file
String electionType = first line of file scanned
If electionType == "IR Election"
    IR_election election = new IR_Election(scan)
else if electionType == "CPL Election"
    CPL_election election = new CPL_Election(scan)
else
    Print("error")
    Return -1
election.run()
```

<u>Methods for Election</u>

Election(electionFile:Scanner, fileName)

```
    typeElection = first line of file
    numBallots = 0;
    this.fileName = fileName
```

coinToss(tieFolk: int[]) : int

```
    winner = getRandomNumber from 0 to tieFolk.size
    Write to audit using audit object
    return tieFolk[winner]
```

shuffleBallots(ballotList:IR_Ballot | CPL_Ballot, num:int): void

```
    Swap ballots randomly for num times
    return
```

setElectionType(typeElection:string): void

```
    this.typeElection = typeElection
    return
```

setNumBallots(numBallots:int):void

```
    this.numBallots = numBallots
    return
```

setFileName(fileName:string): void

```
    this.fileName = fileName
    return
```

setElectionFile(scan:Scanner): void

```
    electionFile = scan
    return
```

getNumBallots():int

```
    return numBallots
```

getFileName():String

```
    return fileName
```

getElectionFile(): Scanner

```
    return electionFile
```

<u>Methods for IR_Election</u>

IR_Election(electionFile:Scanner)

```
    auditIR = new IR_Audit
    Candidates = {}
    numCandidates = 0
    numRemainingCandidates = 0
    currentBallots = {}
    currentBallotCount = 0
    numBallots = 0
    typeElection = “IR Election”
    this.electionFile = electionFile
```

readIRHeader(): void

```
    numCandidates = scan second line of file
    remainingCandidate = numCandidates
    int i from 0 to numCandidates
        candidates[i].ballotCount = 0
        candidates[i].ballots = {}
        Candidates[i].name = scan third line read name
        Candidates[i].party = scan next word without comma
    numBallots = scan next line
```

readIRBallots(): void

```
    Int i from 0 to numBallots
        IR_ballot ballot
        ballot.rank = 0
        ballot.ballotNum = i
        ballot.form = Read nextline from file
        for each character in form
            int position = 0
            if character == ‘,’
                position++
            else
                Treat character as an int
                candidateRank[character] = position
    return
```

findLowestCandidate(): void

```
    int remove = 0;
    while (candidate[remove] == NULL)
         remove++
    lowestVote = candidate[remove].ballotCount
    for i = 0 to numCandidates
        If (candidate[i] != NULL && candidate[i].ballotCount < lowestVote)
            lowestVote = candidates[i].ballotCount
            remove = i
    int lowestCandidates[];
    For i = 0 to numCandidates
        If (candidate[i].ballotCount == lowestVote)
            lowestCandidates.add i
    If (lowestCandidates.size >= 2)
```

```
        remove = coinToss(lowestCandidates)
    return remove
```

removeLowestCandidate(lowest: int): void

```
    for int i = 0 to candidate[i].ballotCount
        candidate.ballots[i].rank++
    initalBallots = candidate[i].ballots
    currentBallotCount = candidate[i].ballotCount
    Candidate[i] = NULL
    numRemainingCandidates -= 1
    return
```

runPopularity(): int

```
    int winner = 0;
    while (candidate[winner] == NULL)
        winner++
    int highestVote = candidate[winner]
    for int i = (winner+1) to numCandidates
        If (candidate[i] != NULL && candidate[i].ballotCount > highestVote)
            highestVote = candidates[i].ballotCount
            winner = i
        else if (candidate[i] != NULL && candidate[i].ballotCount == highestVote)
            Winner cointToss({winner, i}
    return winner

checkMajoirity(): int
    If (numRemainingCandidates <= 2)
        Return runPopuularity()

    For int i = 0 to numCandidates
        If (candidate[i] != NULL && candidate[i] > BallotCount/2)
            Return i
    Return -1

allocateBallots(): void
    For int i = 0 to currentBallotCount
        Ballet giving = currentBallots[i]
        candidate[giving.rank].ballots.add(giving)
        candidate.ballotCount++

run():void
    readIRHeader()
    readIRBallots()
    shuffleBallots()
    While true
        allocateBallots()
        int winner = checkMajority()
            If winner != -1
                Write to audit using audit object
                Display winner using audit object
                return
            If (numRemainingCandidates <= 2)
                    winner = runPopuularity()
                    Write to audit
                    Display winner
                    Return
            int lowest = findLowestCandidate
            removeLowestCandidate(lowest)
            write to audit using audit object
    return
```

getRemainingCandidates(): int

```
    return remainingCandidates
```

getCurrentBallotCount(): int

```
    return currentBallotCount
```

getBallots(): IR_Ballot[]

```
    return currentBallots
```

```
getCandiadates(): Candidate[]
        return candidates

setNumCandidates(num:int): void
        numCandidates = num
        return

setNumRemainingCandidates(num:int): void
        numRemainingCandidates = num
        return

setCurrentBallotCount(num:int): void
        currentBallotCount = num
        return

setCandidates(candidates:Candidate[]):void
        this.candidates = candidates
        return

setCurrentBallots(ballots:IR_Ballot[]):void
        currentBallots = ballots
```

```
    return
```

<u>Candidate Class methods</u>

```
Candidate()
    ballotCount = 0
    ballots = {}
    name = “”
    party = “”

getParty():string
    return party
setParty(party:string): void
    this.party = party
    return

getName(): string
    return name

setName(name:string): void
    this.name = name
    return
getBallotCount():int
```

```
    return ballotCount
```

setBallotCount(count:int): void

```
    ballotCount = count
```

getBallots(): IR_Ballot[]

```
    return ballots
```

setBallots(ballots:IR_Ballots): void

```
    this.ballots = ballots
```

<u>IR_Ballot methods</u>

IR_Ballot()

```
    ballotNum = -1
    rank = -1
    form = “”
    candidateRanking = {}
```

getNumBallot(): int

```
    return ballotNum
```

setBallotNum(num:int): void

```
    ballotNum = num
```

getRank(): int

```
    return rank
```

setRank(num:int): void

```
    rank = num
```

getForm(): string

```
    return form
```

setForm(form:string)

```
    this.form = form
```

getCandidateRankings():int[]

```
    return candidateRanking
```

setCandidateRankings(candidateRanking:int[]): void

```
    this.candidatRanking = candidateRanking
```

<u>Methods for CPL_Election()</u>

readCPLHeader(): void

```
    numParties = scan second line of file
    int i from 0 to numParties
        parties[i].ballotCount = 0
        parties[i].ballots = {}
        parties[i].name = scan party name delimited by comma
    int i from 0 to numParties
        parties[i].candidateList = scan line and split by comma delimiter
    availableSeats = scan integer from line
    numSeats = availableSeats
    numBallots = scan integer from line
```

readCPLBallots(): void

```
    int i from 0 to numBallots
        initialBallots[i] = new CPL_Ballot()
        initialBallots[i].setNumBallots(i)
        initialBallots[i].setPartyNum(split next line by commas, select index with integer
    element 1)

    shuffleBallots(initialBallots, numBallots)
    int i from 0 to numBallots
        parties[initialBallots[i].getPartyNum()].addBallot(initialBallots[i])
```

setFirstSeatsAllocation(): ArrayList<Party>

```
    ArrayList<Party> partiesAvailable
    quota = initialBallots.length / numSeats
    int i from 0 to numParties
        seatsAllocated = parties[i].getBallotCount() / quota  # must be integer division
        if seatsAllocated > parties[i].getNumCandidates()
            parties[i].setSeatsWon(parties[i].getNumCandidates())
        else
            parties[i].setSeatsWon(seatsAllocated)
            parties[i].setRemainder(parties[i].getBallotCount() mod quota)
            if parties[i].getRemainder() > 0
                partiesAvailable.add(parties[i])
        availableSeats -= parties[i].getSeatsWon()
        Write number of seats allocated to party in first round  to audit
```

setRemainderSeatsAllocation(partiesAvailable:ArrayList<Party>): void

```
    Write number of ballots remaining per party to audit
    partiesAvailable.sort() # sort in descending order
    int i from 0 to partiesAvailable.length
        if availableSeats <= 0
            break
        if partiesAvailable[i].getRemainder() == partiesAvailable[i+1].getRemainder()
            coinToss(generate all parties from i to n in partiesAvailable with same remainder)
            i = n + 1
        availableSeats - -
    while availableSeats > 0
        parties[coinToss(generate list of party indices from partiesAvailible)].incrementSeatsWons()
        availableSeats - -
```

run(): void

```
    readCPLHeader()
    readCPLBallots()
    write header for audit, including election type, parties and their ballots, number of total ballots,  number of seats to be allocated quota
    setRemainderSeatsAllocation(setFirstSeatAllocations())
    Write total seats allocated per party to audit
    write election winners to audit file
```

<u>Methods for Party</u>

Party()

```
    candidateList = “”
    numCandidates = 0
    seatsWon = 0
    ballotCount = 0
    Ballots = {}
    Remainder = 0
```

getSeatsWon(): int

```
    return seatsWon
```

setSeatsWon(seatsWon:int): void

```
    this.SeatsWon = seatsWon
```

getNumCandidates(): int

```
    return numCandidates
```

setNumCandidates(numCan:int): void

```
    numCandidates = numCan
```

getBallotCount(): int

```
    Return ballorCount
```

setBallotCount(num: int): void

```
    ballotCount = num
```

addBallot(ballot:CPL_Ballot): void

```
    ballotsCount++
    ballots.add(Ballot)
    return
```

getRemainder(): int

```
    return remainder
```

setRemainder(remainder:int): void

```
    This.remainder = remainder
```

incrementSeatsWon(): void

```
    seatsWon++
```

<u>Methods for CPL_Ballot</u>

CPL_Ballot()

```
    partyNum = -1
    ballotNum = -1
```

getPartyNum(): int

```
    return partyNum
```

setPartyNum(num:int): void

```
    partyNum = num
```

getNumBallots(): int

```
    return ballotNum
```

setNumbBallot(num:int): void

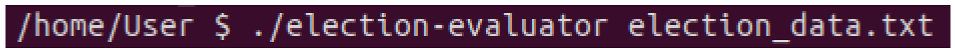
ballotNum = num

6. Human Interface Design

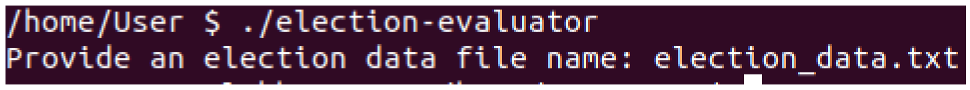
6.1 Overview of User Interface

The system requires little user input or interaction. All interaction will happen with a text based interface. The user can either provide an election data file name to the system via a command line argument or provide it via a prompt during runtime. The system will then process the election data as specified in the election data file and display the results of the election to the terminal. An audit file will be generated for the user in the same current working directory as the system is run.

6.2 Screen Images



```
/home/User $ ./election-evaluator election_data.txt
```



```
/home/User $ ./election-evaluator
Provide an election data file name: election_data.txt
```

The user will enter in information using the terminal where they will start up the program and be prompted from the screen to enter information or to directly enter in the program and file name on one line.

```
CPL Election Audit

Parties
Party 1 : candidate 1, candidate 2, ...
Party 2 : candidate 1, candidate 2, ...
Party 3 : candidate 1, candidate 2, ...
Party 4 : candidate 1, candidate 2, ...

Number of Seats to be Allocated: number of seats to be allocated
Total Vote Count : total vote count number
Quota : quota for seat

Party Votes
Party 1 : initial vote number
Party 2 : initial vote number
Party 3 : initial vote number

Initial Seat Allocation
Party 1 : initial seat allocation number
Party 2 : initial seat allocation number
Party 3 : initial seat allocation number

Party Votes Remaining
Party 1 : remaining votes number
Party 2 : remaining votes number
Party 3 : remaining votes number

Remainder Seat Distribution
Party 1 : remainder seat allocation number
Party 2 : remainder seat allocation number
Party 3 : remainder seat allocation number

Total Seat Distribution
Party 1 : total seat allocation number
Party 2 : total seat allocation number
Party 3 : total seat allocation number

Election Winners
Party 1 : Winning Candidate 1, Winning Candidate 2. ...
Party 2 : Winning Candidate 1, Winning Candidate 2. ...
Party 3 : Winning Candidate 1, Winning Candidate 2. ...
Party 4 : # any party that does not win any seats will be left blank
```

```
IR Election Audit

Candidates
Candidate 1
Candidate 2
Candidate 3

Total Ballot Count :  total ballot count

Party Votes
Party 1 : initial vote number
Party 2 : initial vote number
Party 3 : initial vote number

Vote Allocation Round 1
Candidate 1 : current candidate rank votes
Candidate 2 : current candidate rank votes
Candidate 3 : current candidate rank votes

Vote Allocation Round 2
Candidate 1 : current candidate rank votes
Candidate 2 : current candidate rank votes
Candidate 3 : current candidate rank votes

Vote Allocation Round 3
Candidate 1 : current candidate rank votes
Candidate 2 : current candidate rank votes
Candidate 3 : current candidate rank votes

# vote allocation rounds continue until only one candidate left

Election Winner
Winning Candidate
```

Additionally, after each candidate's name, the list of ballet numbers that each candidate got will be printed out. Whenever a tie takes place, the audit will print out who tied and who won or lost the tie.

6.3 Screen Objects and Actions

The system has no screen objects or actions

7. Requirements Matrix

Use Case ID	**Description**	**System**
UC_4.1	System reads in CSV file	This will happen in the Election class, the file will be read in and stored in the electionFile attribute.
UC_4.3	Determine Election Type	This will happen in the setElectionType() method in the Election class.
UC_4.6	Determine the number of ballots	This will happen the in the setNumBallots() method in the Election class
UC_4.7	Determine number of seats	This is done in the getNumSeats() method in CPL_Election class
UC_4.8	Read IR Ballots	This is done in the readIRBallots() method in IR_Election class
UC_4.9	Read CPL Ballots	This is done in the readCPLBallots() method in the CPL_Election class
UC_4.10	Shuffle Ballots	This is done in the shuffleBallots() method in the Election class
UC_4.11	Run IR Election	This is done in the run() method in the IR_Election class
UC_4.12	Run CPL Election	This is done in the CPL_Election class
UC_4.13	Coin Toss	This is done in the coinToss() method in the Election class
UC_4.16	Produce IR Audit File	This will happen in the writeToFile() method in the IR_Audit class, returnAudit()

		will return the file
UC_4.17	Produce CPL Audit File	This will happen in the writeToFile() method in the CPL_Audit class, returnAudit() will return the file
UC_4.18	Run Popularity	This will happen in the runPopularity() method in the IR_Election class

8. Appendices

Appendix A:

CPL voting algorithm works as follows:

- Voters indicate their preference for a particular party on their ballot and the parties then receive seats in approximate proportion to their share of the total vote.
- Algorithm to determine seat allocations:
  The quota is calculated (see quota)
- For the first round of seat allocations, seats are allocated by whole numbers chunks of the quota size of each party's votes. For example, if the quota is 10,000 and a party receives 32,000 votes, it will receive 3 seat allocations for the first round.
- For the second round of seat allocations, the votes that did not contribute to the first round seat allocation are considered. Considering the example from the first round, the votes considered will be 2,000 votes. The seats not allocated in the first round will be allocated to the parties in descending order of remaining votes after the first allocation.
- The party will receive seats that are a sum of those received from the first and second rounds of allocations. The party will then choose to elect candidates that are sorted in a list of priority to be elected. Those with a higher priority will receive an allocated party seat over those with a lower priority. That is, the candidates have a particular order in which seats are allocated if their party wins.

Appendix B:

IR Voting algorithm works as follows:

- Voters can rank all the candidates on their ballot based on their preference of who they want to win the election. These rankings range from 1 to the number of candidates.
- Algorithm to determine winner:
- The candidates' votes are set to be the number of first place votes they each received.
- If no candidate holds the majority of the vote, the candidate with the least amount of votes is eliminated as a potential winner.
- The votes for the second rank is then added to the votes of each candidate. If no candidate holds a majority, another candidate is eliminated in a similar manner and this process continues until a candidate reaches a majority of the votes.

- If a candidate ever reaches a majority of the votes, they are declared the winner
- If a candidate never reaches a majority after all the rank votes are distributed, the candidate with the plurality wins.