

Team 5  
**Voting System**  
Software Design Document

Name (s):

Camden Fessler (fessler022)

Ethan Johnson (john18919)

Aaron Raines (raines124)

Maria Zavala (zavala054)

Lab Section: Workstation:

Date: 02/21/2023

---

## TABLE OF CONTENTS

<b>1. INTRODUCTION</b>	<b>2</b>
1.1 Purpose	2
1.2 Scope	2
1.3 Overview	2
1.4 Reference Material	3
1.5 Definitions and Acronyms	3
<b>2. SYSTEM OVERVIEW</b>	<b>3</b>
<b>3. SYSTEM ARCHITECTURE</b>	<b>4</b>
3.1 Architectural Design	4
3.2 Decomposition Description	5
3.3 Design Rationale	8
<b>4. DATA DESIGN</b>	<b>9</b>
4.1 Data Description	9
4.2 Data Dictionary	11
<b>5. COMPONENT DESIGN</b>	<b>20</b>
<b>6. HUMAN INTERFACE DESIGN</b>	<b>26</b>
6.1 Overview of User Interface	26
6.2 Screen Images	27
6.3 Screen Objects and Actions	28
<b>7. REQUIREMENTS MATRIX</b>	<b>29</b>
<b>8. APPENDICES</b>	<b>31</b>

## **1. INTRODUCTION**

### **1.1 Purpose**

This Software Design Document provides the design details of a Voting System capable of performing IR and CPL voting. The expected audience are the voting officials, developers, and testers of the system.

### **1.2 Scope**

The Voting System is software designed to calculate the number of votes cast for each candidate in an election, generate reports on the results of the election, and determine the winner. The system will be developed using Java programming language and is expected to operate on CSE lab machines. The system is designed to be used by Election Officials, Auditors, Developers, Programmers, and Testers, who will have access to different features and tools of the system based on their roles and responsibilities. The system functions include data processing, data reporting, audit trail, and user interface. The data processing function is responsible for calculating the number of votes cast for each candidate or party. The data reporting function generates reports on the results of the election, including the number of votes cast for each candidate or party and the winner of the election. The audit trail function records all system activities for future reference. The user interface function allows users to interact with the system. The benefits of this system include its consistency, accuracy, and speed in providing results.

### **1.3 Overview**

The remaining chapters and their contents for this Software Design Document for the Voting System provides a comprehensive overview of the design of the system, its architecture, and its various components. The system overview section provides an overview of the software, including its functions, user classes, and benefits. The system architecture section details the software's architectural design, decomposition description, and design rationale. The data design section explains the data description and data dictionary. The component design section explains the design of the various software components, including their roles and interactions. The human interface design section describes the user interface of the system, including screen images, screen objects, and actions. The requirements matrix section provides a summary of the system's requirements and how they are fulfilled by the software components. Finally, the appendices section includes additional information that

---

supports the design of the system. Overall, the document provides a detailed guide to the software design and implementation of the Voting System.

## 1.4 Reference Material

CSci 5801: Software Engineering I, Spring 2023, Project 1 – Waterfall Methodology, Software Design Document (SDD) for Voting System. The University of Minnesota.

Zavala, M., Raines, A., Fessler, C., Johnson, E. (2023). Software Requirements Specification for Voting System Version 1.0 approved. The University of Minnesota. February 3, 2023.

## 1.5 Definitions and Acronyms

Term	Definition
Algorithm	A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
IR/IRV	Instant Runoff voting
CPL	Closed party list voting
Data structure	A storage format that is usually chosen for efficient access to data.
CSV	A file that contains comma-separated values.
NULL	having no legal or binding force; invalid.
UC	User Case
Command Line	This provides a means of setting parameters for invoking a program and providing information to it as to what actions it must perform.

## 2. SYSTEM OVERVIEW

The voting system software aims to process and calculate the votes cast for candidates and/or parties in an election. The system is designed to be used by Election Officials, Auditors, Developers, Programmers, and Testers. The software is capable of processing two types of voting: Instant Runoff Voting (IR) and Closed Party List (CPL) voting. The system assumes that all CSV files are cleaned and formatted to a specified preference. The system will establish an election winner and generate reports on the results of the election. The system is a follow-on member of a product family designed to automate the process of counting votes in an election.

---

The major functions that the vote-counting system must perform are data processing, data reporting, audit trail, and user interface.

### **3. SYSTEM ARCHITECTURE**

#### **3.1 Architectural Design**

##### **Subsystems:**

**Election:** When the election is called, this subsystem is the one that is started. The election class is what initially sets up the election and is in charge of every other subsystem in the program. This is responsible for reading the election file initially, parsing through all of the information, and saving it for later. It will also save all ballots from the election file into their own Ballot data type. The election subsystem is also responsible for creating an Instant Runoff Election system or a Closed Party List Election system according to the initial file. Throughout all of this, it will also be collecting information with the help of an Audit Info Collector.

**Instant Runoff:** This subsystem is used specifically when the Election class reads the election file and determines that an Instant Runoff election is being called. This subsystem will be responsible for taking in a list of Ballots and Candidates and will determine the winner of the election according to IR standards. This will include dropping candidates where needed, redistribution of votes, and deciding the winner of a tie. By the end of the election, this subsystem will have one Candidate as a winner for the Election class and Audit Info Collector class to use for the audit file.

**Closed Party List:** This system is used specifically when the Election class reads the election file and determines that a Closed Party List Election is being called. This subsystem will be responsible for taking in a list of Ballots and Parties and will determine the winners of the election according to CPL standards. This will include assigning seats to candidates as well as determining the winner of a tie fairly. By the end of the election, this subsystem will have a list of Candidates as winners for the Election class and Audit Info Collector class to use for the audit file.

**Party:** This class will act as a data structure used in elections and will be responsible for holding a list of Candidates assigned to that party. The Party class will hold the name of the party, a list of Candidates, the number of popular votes that the party has received, the ballots they received, and the number of remaining votes from CPL.

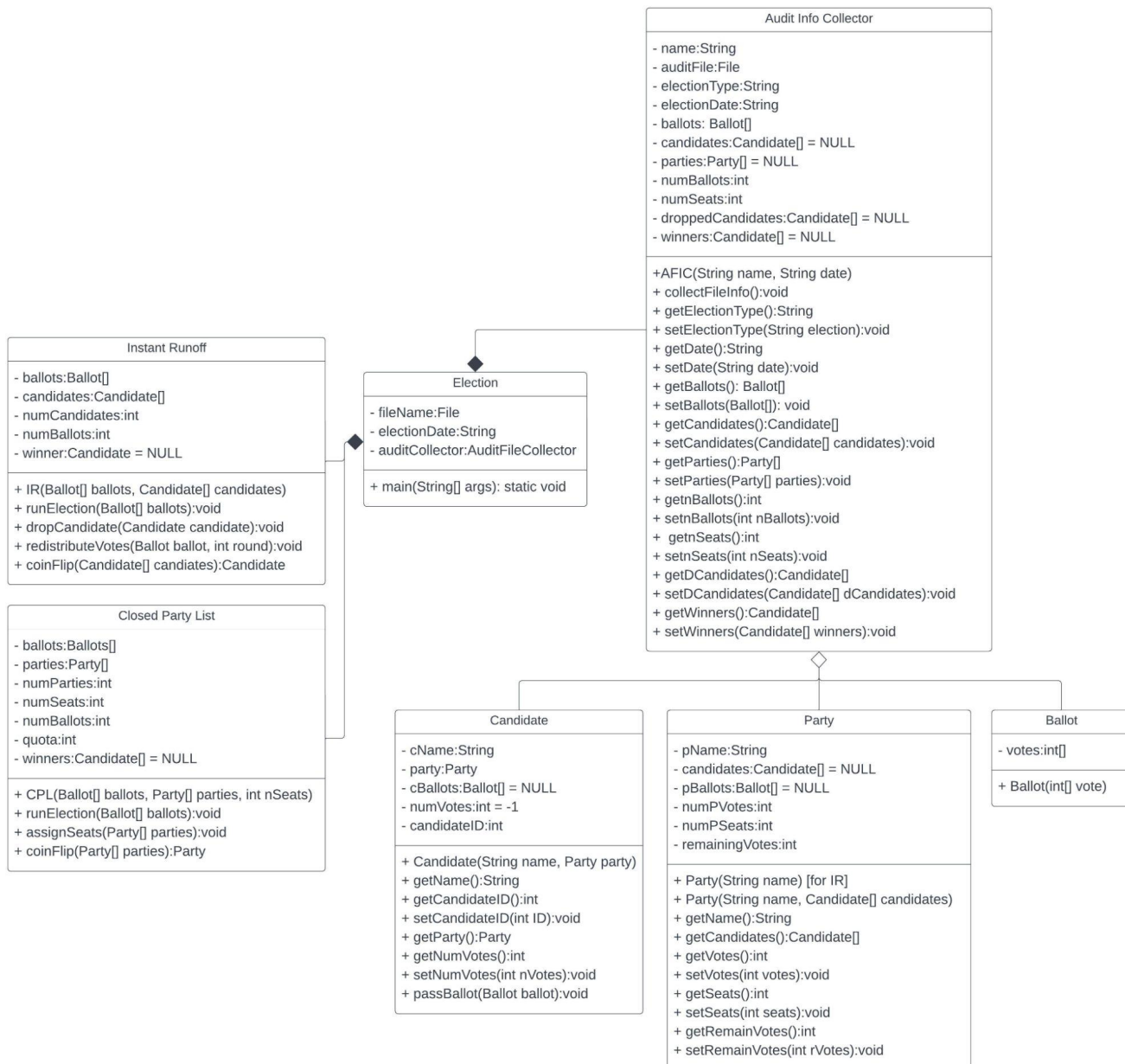
**Candidate:** This class will act as a data structure used in elections and will be responsible for holding the information of a specific candidate. This information includes the name of the Candidate, the Party of that candidate, and the number of votes and ballots they have received.

**Ballot:** This class will act as a data structure used in elections and will be responsible for holding the information of a single ballot obtained from the election file. It will hold an int list symbolizing the votes of the ballot.

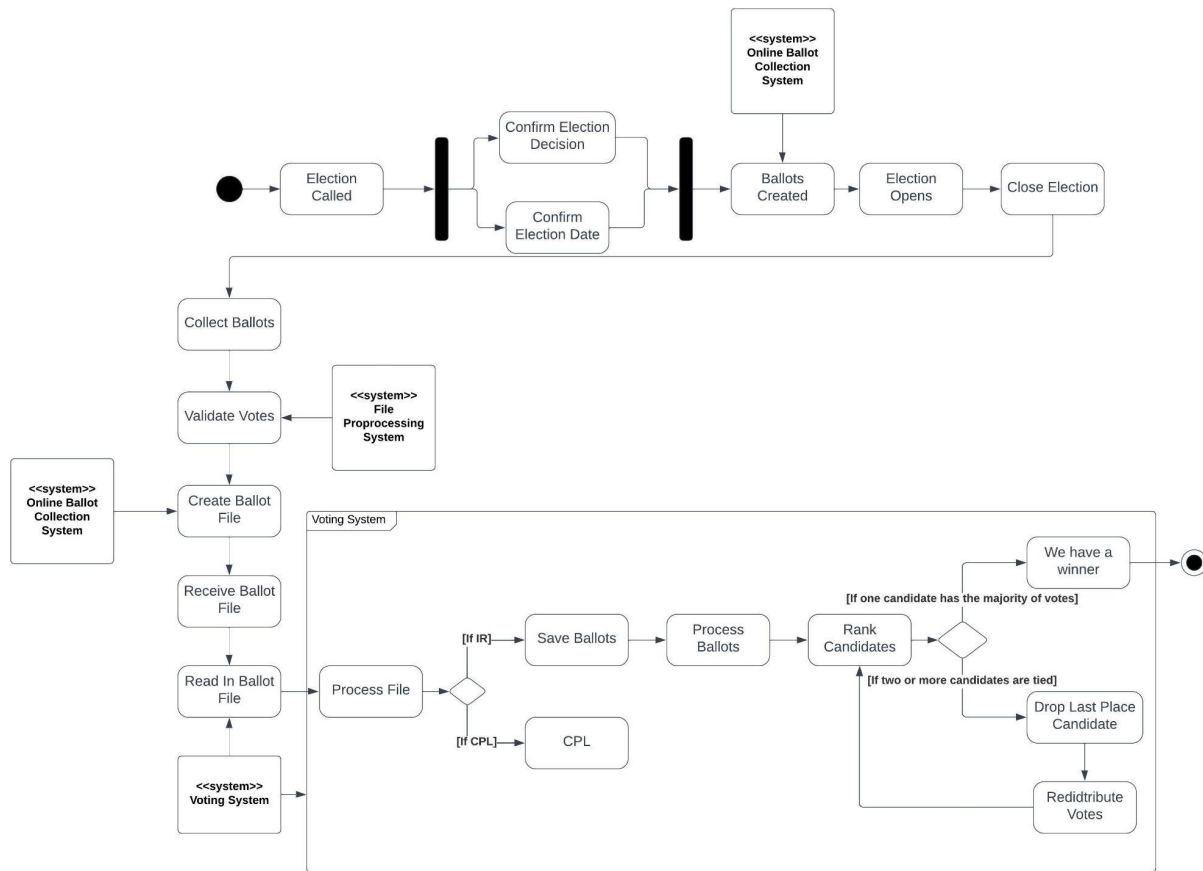
**Audit Info Collector:** This class will be responsible for collecting all of the necessary information needed for the audit file after the election has concluded. It will be instantiated from

the beginning by Election and will have its functions called when information needs to be updated. At the end of the program, this subsystem is used to gather all information for the audit file and write it to the text file.

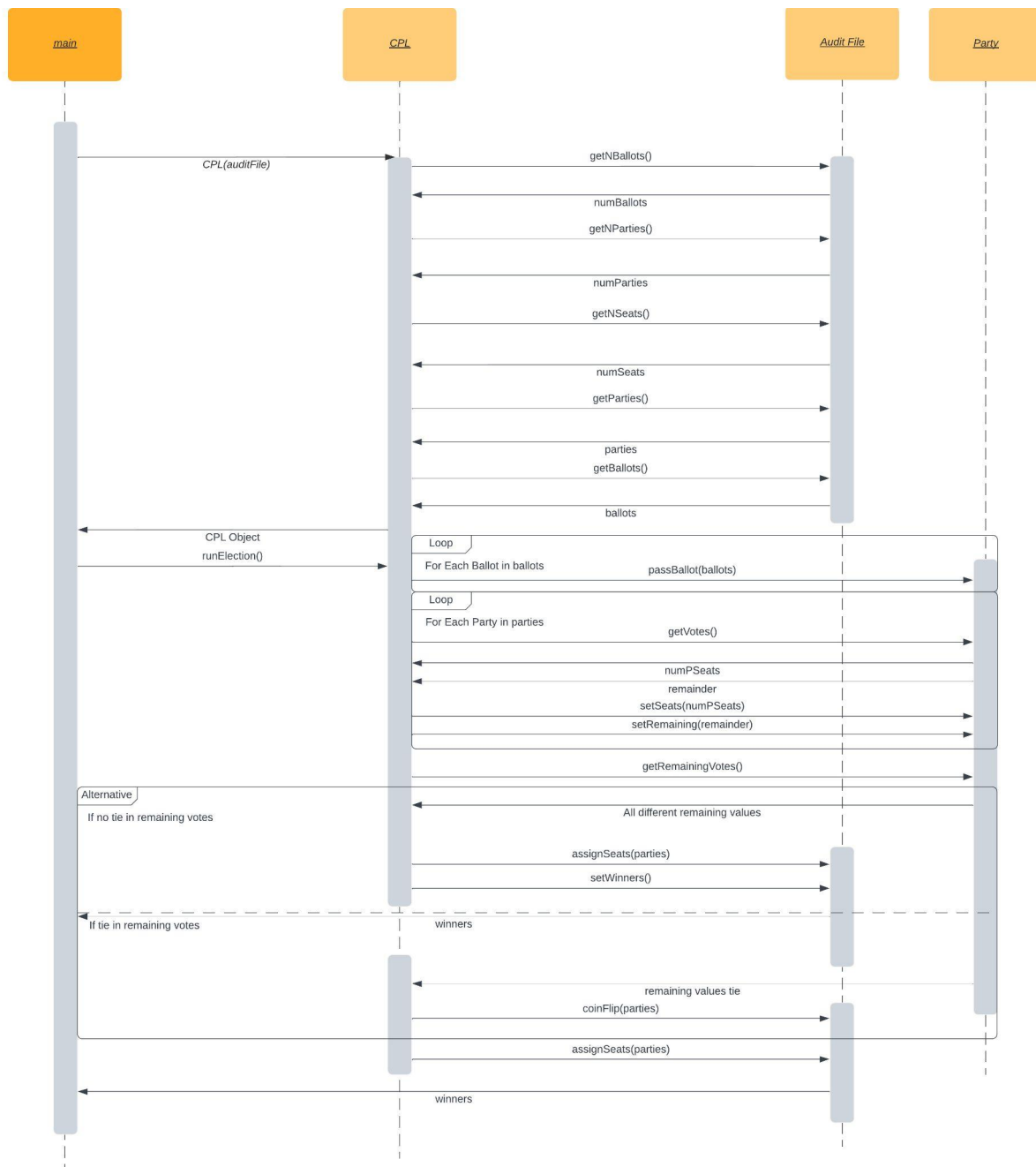
### 3.2 Decomposition Description



Above is the class diagram with each block being a class in the Voting system with their objects and functions listed inside and their inheritance displayed through the lines connecting them.



This Activity Diagram starts when an Election is initially called. After the election is called, the election is confirmed and the election date is set. After these processes have been completed, the Online Ballot Collection System creates online ballots for voters to use. The election then opens, voters will cast their ballots, and the election then closes. The votes will be collected, validated, preprocessed, and made into a file. That file is received by the voting system which processes the file determining the election type. It then processes the ballots in the file and ranks the candidates. If there is a tie, a candidate is dropped, the votes are redistributed, and candidates are ranked again. This will continue until a winner is declared by the system.



This Sequence Diagram above shows interactions between system components. It starts with the main calling the CPL function affecting the CPL object. This calls all the functions associated with the CPL class which fills the Audit file object with information about the election. The CPL object then enters two different loops, one calling a function to fill the Party object with ballot information and the other calling a function to fill the Party object with votes information. Both loops then return back to the CPL object information about seats. Then if there is no tie a function to assign seats is called and a function setting the winner(s) is called assigning the winner and seats to the audit file object. If there is a tie, the Party object



---

gives the remaining values to the CPL object which then calls a function to flip a coin to determine winners. Then a function is called by CPL again to assign seats and set winners to the audit file object which in turn delivers the winner's information back to the main.

### **3.3 Design Rationale**

The rationale behind our design choice was simplicity for the user to understand as well as simplicity for the program itself. Having one main class, Election, be in charge of calling the different forms of elections is the simplest way of implementing this idea because if any more election types needed to be added, it would be very simple. Having the two election types as their own classes is also extremely important because they have different standards and methods of determining the winner of elections. Including the data structures Candidate, Party, and Ballot also simplifies the process of running the election because once this information is collected by Election, it can be passed into any election type as necessary, since they all work with the same data structures. Finally, including an Audit Info Collector is very useful because the audit file contains a lot of information, and having a place to store and consolidate this information makes retrieving it for the audit file much simpler and more efficient than having to backtrack through the election process.

## 4. DATA DESIGN

### 4.1 Data Description

**Audit Info Collector:** The audit file class is the way we store the information about the election by parsing through the CSV file. As well as running the election algorithms. We get the following information:

Attribute Name	Attribute Type	Description
name	String	Store file name Passed to audit constructor
electionType	String	Store Election type: IR or CPL Retrieved from the 1st line in CSV.
electionDate	String	Store the day of the election being processed. Input by the user when prompted.
auditFile	File	Store a file object for audit Create with name + date
ballots	Ballot[]	Store a collection of ballots parse through CSV file
candidates	Candidate[ ]	Store the names of the candidates in elections. IR: Retrieved from the 3rd line in CSV. CPL: Retrieved from 4th line to the number of parties in CSV.
parties	Party[ ]	Store the names of the parties in the CPL election. CPL: Retrieved from the 3rd line in CSV.
numBallots	int	Store the number of ballots in the elections. IR: Retrieved from 4th line in CSV. CPL: Retrieved from the line: number of parties + 5
numSeats	int	Store the number of seats in the CPL election. CPL: Retrieved from the line: number of parties + 4
droppedCandidates	Candidate[ ]	Store the candidates that were dropped in the IR algorithm. IR: Each time we run our IR algorithm if a candidate has been dropped it will be added.
winners	Candidate[ ]	Store the candidate(s) who are considered winners of elections. IR: Retrieve the result of running the IR algorithm with the getWinner() method. CPL: Retrieve the result of running the CPL algorithm with the getWinner() method.

**Ballot:** The ballot class will be storing each ballot in elections from the CSV file. For CPL the ballot will contain what party each voter has voted for. For IR the ballot will contain the rank number each voter has given to each candidate.

Attribute Name	Attribute Type	Description
votes	int[ ]	IR: Store array of rankings for each vote Retrieve from CSV file parsing each line starting at line 5. CPL: Store an array of numbers for each vote Retrieve from the CSV file parsing each line starting from the number of parties + 6

**Candidate:** The candidate class will store the information of each candidate including the following information.

Attribute Name	Attribute Type	Description
cName	String	Store the name of the candidate IR: Created from parsing CSV in the 3rd line. CPL: Created from parsing CSV in the 4th line to the number of parties.
party	Party	Store the party each candidate belongs to IR: Retrieve from the letter in ( ) following the names of candidates in CSV. CPL: Indicated by the line number each candidate is listed on CSV matched to the number each party is represented by.
numVotes	int	Store the number of votes each candidate has received for an IR election. Retrieve from counting the number of cBallots received.
cBallots	Ballot[ ]	Store the ballots each candidate has received in the IR election. IR: Assigned from running the IR algorithm
candidateID	int	Store a unique number representing the position of the candidate in vote IR: get from the position in CSV file

**Party:** This class will store the party information for the CPL election.

Attribute Name	Attribute Type	Description
pName	String	Store the name of the party. CPL: Created from parsing CSV in the 3rd line.
candidates	Candidate[ ]	Store the candidates that are under each party for the CPL election. CPL: Indicated by the line number each candidate is listed on CSV matched to the number each party is represented by.
pBallots	Ballot[ ]	Store the ballots each Party has received in the CPL election. IR: Assigned from running the CPL algorithm
numPVotes	int	Store the number of votes received by each party. CPL: Retrieve after running the CPL algorithm.
numPSeats	int	Store number of seats given to each party CPL: Retrieve after running the CPL algorithm
remaningVotes	int	Store the number of remaining votes CPL: Determined by running the CPL algorithm after dividing the number of votes received by the quota which is the number of total votes divided by seats.

## 4.2 Data Dictionary

### Audit File Information Collector

**Name:** AuditInfoCollector

**Type:** Information Collector / Database

**Description:** This class will collect information about each election, as well as calculations made in each election.

**Attributes:**

name: String

auditFile: File

electionType: String

electionDate: String

ballots: Ballot[]

candidates: Candidate[ ]

parties: Party[ ]

---

numBallots: int  
numSeats: int  
droppedCandidates: Candidate[ ]  
winners: Candidate[ ]

**Constructors:**

Name: AFIC( ) Arguments: name: String date: String
---

**Methods:**

Name: collectFileInfo()  
Arguments: None  
Return: void

Name: getElecType( )  
Arguments: None  
Return: String

Name: setElecType( )  
Arguments:  
    elec: String  
Return: void

Name: getDate( )  
Arguments: None  
Return: String

Name: setDate( )  
Arguments:  
    date: string  
Return: void

Name: getBallots()  
Arguments: None  
Return: Ballot[]

Name: setBallots( )  
Arguments: Ballot[]  
Return: void

Name: getCandidates( )  
Arguments: None  
Return: Candidate[ ]

Name: setCandidates( )

---

Arguments:  
Candidates: Candidate[ ]  
Return: void

Name: getParties( )  
Arguments: None  
Return: Party[ ]

Name: setParies( )  
Arguments:  
parties: Party[ ]  
Return: void

Name: getnBallots( )  
Arguments: None  
Return: int

Name: setnBallots( )  
Arguments:  
ballots: int  
Return: void

Name: getnSeats( )  
Arguments: None  
Return: int

Name: setnSeats( )  
Arguments:  
seats: int  
Return: void

Name: getDCandidates( )  
Arguments: None  
Return: Candidate[ ]

Name: setDCandidates( )  
Arguments:  
dCandidates: Candidate[ ]  
Return: void

Name: getWinners( )  
Arguments: None  
Return: Candidate[ ]

Name: setWinners( )  
Arguments:  
winners: Candidate[ ]

---

Return: void

---

---

## Ballot

---

**Name:** Ballot

**Type:** Datatype

**Description:** This will hold information about each ballot encountered in CSV files.

**Attributes:**

votes: int[]

**Constructors:**

Name: Ballot( ) Arguments: votes: int[]
---

**Methods:** None

---

## Candidate

---

**Name:** Candidate

**Type:** Datatype

**Description:** This class will hold information about each candidate in the election.

**Attributes:**

cName: String

party: Party

cBallot: Ballot[ ]

numVotes: int

candidateID: int

**Constructors:**

Name: Candidate( ) Arguments: name: string party: Party
--

**Methods:**

Name: getName( )

Arguments: None

---

Return: String

Name: getCandidateID()

Arguments: None

Return: int

Name: setCandidateID()

Arguments:

ID: int

Return: void

Name: getParty()

Arguments: None

Return: Party

Name: getVotes()

Arguments: None

Return: int

Name: setVotes()

Arguments:

votes: int

Return: void

Name: passBallot()

Arguments:

ballot: Ballot

Return: void

---

### Closed Party List

---

**Name:** ClosedPartyList

**Type:** Algorithm

**Description:** This class will collect information about the CPL election and will perform the defined CPL algorithm to determine seats for winners.

**Attributes:**

ballots: Ballots[ ]

parties: Party[ ]

numBallots: int

numParties: int

numSeats: int



---

quota: int

winners: Candidate[ ]

**Constructors:**

Name: CPL( )
--------------

Arguments:
------------

electionFile: AuditFile
-------------------------

**Methods:**

Name: runElection( )

Arguments: None

Return: void

Name: assignSeats( )

Arguments:

parties: Party[ ]

Return: void

Name: coinFlip():

Arguments:

parties: Party[]

Return: Party

---

## Election

---

**Name:** Election

**Type:** Program

**Description:** This is the main class that runs the entire voting system.

**Attributes:**

fileName: File

electionDate: String

auditCollector: AuditFileCollector

**Methods:**

Name: main( )

Arguments:

args: String[ ]

Return: static void

---

---

## Instant Runoff

---

**Name:** InstantRunoff

**Type:** Algorithm

**Description:** This class will collect information about IR election and will perform the defined IR algorithm to determine a winner.

**Attributes:**

ballots: Ballot[ ]

candidates: Candidate[ ]

numBallots: int

numCandidates: int

winner: Candidate

**Constructors:**

Name: IR( )

Arguments:

electionFile: AuditFile

**Methods:**

Name: runElection( )

Arguments: None

Return: void

Name: dropCandidate( )

Arguments:

candidate: Candidate

Return: void

Name: redistributeVotes()

Arguments:

ballot: Ballot

round: int

return: void

Name: coinFlip( )

Arguments:

candidates: Candidate[ ]

Return: Candidate

---



---

## Party

---

**Name:** Party**Type:** Datatype**Description:** This class**Attributes:**

pName: String  
 candidates: Candidate[ ]  
 pBallots: Ballot[ ]  
 numPVotes: int  
 numPSeats: int  
 remaningVotes: int

**Constructors:**

Name: Party( ) Arguments: name: String	Name: Party( ) Arguments: name: String candidates: Candidate[ ]
--	--

**Methods:**

Name: getName( )  
 Arguments: None  
 Return: String

Name: getCandidates( )  
 Arguments: None  
 Return: Candidate[ ]

Name: getVotes( )  
 Arguments: None  
 Return: int

Name: setVotes( )  
 Arguments:  
     votes: int  
 Return: void

Name: getSeats( )  
 Arguments: None  
 Return: int

Name: setSeats( )  
 Arguments:

---

seats: int  
Return: void

Name: getRemainVotes( )  
Arguments: None  
Return: int

Name: setRemainVotes( )  
Arguments:  
    remainder: int  
Return: void

## 5. COMPONENT DESIGN

### Pseudo code for Audit File Information Collector Methods:

```
Class Name: AuditFile

AuditFile(String name, String date) {
    this.name = name
    this.date = date
    auditFile = Create a file with the name set to fileName + electionDate
}

void collectFileInfo() {
    electionType = first line in file

    if (electionType == IR) {
        read/set numCandidate = line 2
        parse line 3 read/create/set candidates

        read/set numBallots = line 4
        for i = line 5 to numBallots {
            create ballot
            add to ballots
        }
    } else {
        read/set numParties = line 2
        parse line 3 read/create/set parties

        for i = line 4 to numberof parties
            for each name on each line
                read/create/set candidates

        read/set numSeats = line number of parties + 4
        read/set numBallots = line number of parties + 5

        for i = line number of parties + 6 to numBallots {
            create ballot
            add to ballots
        }
    }
}
```

---

```
void setElecType(String election) {
    this.electionType = election
}

String getDate( ) {
    return this.electionDate
}

void setDate(String date) {
    this.electionDate = date
}

Ballot[] getBallots( ) {
    return this.ballots
}

void setElecType(Ballot[] ballots) {
    this.ballots = ballots
}

Candidate[] getCandidates( ) {
    return this.candidates
}

void setCandidates(Candidate[] candidates) {
    this.candidates = candidates
}

Party[] getParties( ) {
    return this.parties
}

void setParies(Party[] parties) {
    this.parties = parties
}

int getnBallots( ) {
    return this.numBallots
}
```

---

```
void setnBallots(int ballots) {
    this.numBallots = ballots
}

int getnSeats( ) {
    return this.numSeats
}

void setnSeats(int seats) {
    this.numSeats = seats
}

Candidate[] getDCandidates( ) {
    return this.droppedCandidates
}

void setDCandidates(Candidate[] dCandidates) {
    this.droppedCandidates = dCandidates
}

Candidate[] getWinners( ) {
    return this.winners
}

void setWinners(Candidate[] winners) {
    this.winners
}
```

## Pseudo code for Candidate Methods:

Class Name: Candidate

```
Candidate(String name, Party party) {
    this.cName = name
    this.party = party
}

String getName( ) {
    return this.cName
}

int getCandidateID() {
    return this.candidateID
}

void setCandiateID(int ID) {
    this.candidateID = ID
}
```

```
Party getParty( ) {
    return this.party
}

int getNumVotes( ) {
    return numVotes
}

void setNumVotes(int nVotes) {
    this.numVotes = nVotes
}

void passBallot(Ballot ballot) {
    Add ballot to cBallot array
}
```

## Pseudo code for Party Methods:

Class Name: Party

```
Party(String name) {
    this.pName = name
}

Party(String name, Candidate[] candidates) {
    this.pName = name
    this.candidates = candidates
}

String getName( ) {
    return this.pName
}

Candidate[] getCandidates( ) {
    return this.candidates
}

int getVotes( ) {
    return this.numPVotes
}
```

```
void setVotes(int votes) {
    this.numPVotes = votes
}

int getSeats( ) {
    return this.numPSeats
}

void setSeats(int seats) {
    this.numPSeats = seats
}

int getRemainVotes( ) {
    return this.remaningVotes
}

void setRemainVotes(int remainder) {
    this.remaningVotes = remainder
}
```



## Pseudo code for Instant Runoff Methods:

Class Name: InstantRunoff

```
IR(AuditFile electionFile) {
    this.numballots = electionFile.getnBallots()
    this.numCandidates = electionFile.getnParties()
    this.candidates = electionfile.getcandidates()
    this.ballots = electionFile.getBallots()
}

void runElection() {
    for each ballot in ballots {
        int ID = get number 1 choice and set that index
        candidates[ID].passBallot(ballot);
    }

    for each candidate {
        compare number of votes recieved and set max votes as winner

        if there is a tie then {
            check for lowest candidate
            if there is a tie between lowest voted {
                do a coin flip to detremine candidate to drop
            } else {
                drop the lowest voted candidate
            }
        }
    }
}

void dropCandidate(Candidate candidate) {
    for each ballot in candidate {
        Redistribute Votes of candidate
    }

    set candidate as a dropped canidate
}

void redistributeVotes(Ballot ballot, int round) {
    look at ballot and select round number from ballot
    add this ballot to candidate who corresponds to round vote
}

Candidate coinFlip(Candidate[] candidates) {
    generate random number
    randomly select candidate from candidates according to random number
}
```

## Pseudo code for Closed Party List Methods:

Class Name: ClosedPartyList

```

CPL(AuditFile electionFile) {
    this.numBallots = electionFile.getnBallots()
    this.numParties = electionFile.getnParties()
    this.numSeats = electionFile.getnSeats()
    this.parties = electionFile.getParties()
    this.ballots = electionFile.getBallots()
}

void runElection() {
    for each ballot in ballots {
        get party choice match to party
        party.passBallot(ballot);
    }

    Determine quota = total number of ballots / total number of seats

    for each party in parties {
        int numPSeats = party.getVotes() / quota
        int remainder = party.getVotes() % quota

        party.setSeats(numPSeats)
        party.setRemainingVotes(remainder)

        if party.getRemainingVotes() for two or more parties are equal...there is a tie {
            coinFlip(Party[] parties)
        }
    }

    assignSeats(Party[] parties)
}

void assignSeats(Party[] parties) {
    for each party in parties {
        for each candidate and i < party.getSeats() {
            add candidate to winners
        }
    }
}

Party coinFlip(Party[] parties) {
    randomly select winner(s) from p
}

```

---

## Pseudo code for Election Methods:

Class Name: Election

```
void main(String[] args) {
    if args is equal to 0 then
        Prompt user to [enter the file name]
        set fileName = input
    else
        Set fileName form args[0]

    Prompt user [enter the date of election]
    electionDate = Save the input

    auditFile = AuiditFile(fileName, date)
    auditFile.collectFileInfo()

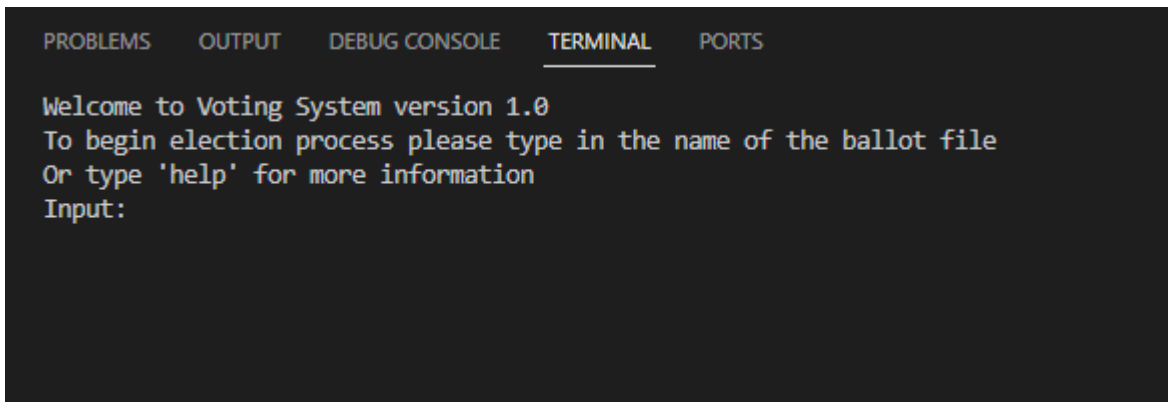
    if auditFile.getElectionType() == IR {
        ir = IR(auditFile)
        ir.runElection()
    }
    else {
        cpl = CPL(auditFile)
        cpl.runElection()
    }
}
```

## 6. HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

Once the user has started the voting system in the command line they will be greeted with a welcome message. Following the message, the user will be able to input an election file or ask for help. The help feature will tell the user what the voting system is and how to use it. If the user inputs a file name the system will begin to process it. Meanwhile, the system may ask the user questions about the file to get missing information or to speed up processing time. After the file has been processed and the voting algorithm has been finished the winner of the election will be printed on the screen for the user. The system will continuously ask for more input until the user quits the system.

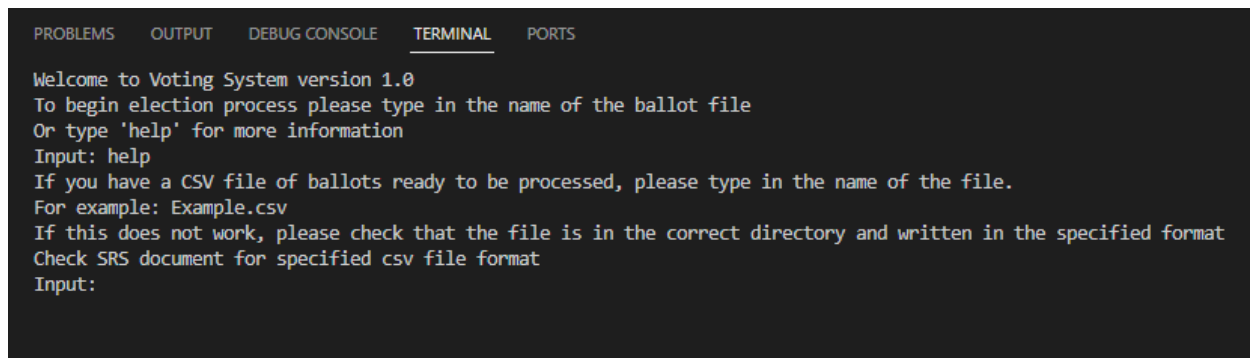
## 6.2 Screen Images



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input:
```

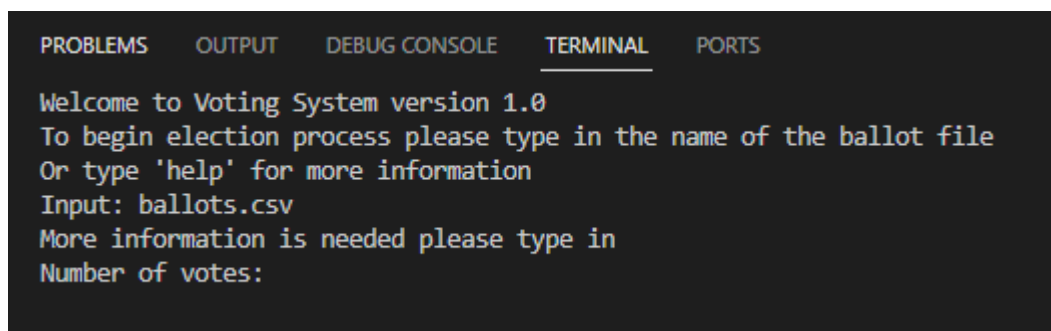
Inputting “Help” will print all possible input commands to the screen



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

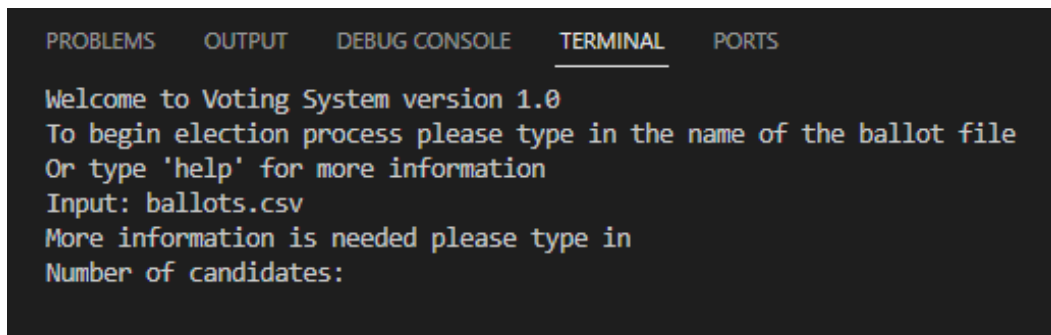
Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: help
If you have a CSV file of ballots ready to be processed, please type in the name of the file.
For example: Example.csv
If this does not work, please check that the file is in the correct directory and written in the specified format
Check SRS document for specified csv file format
Input:
```

If the user inputs a CSV file, the file will be read and will prompt the user for any more needed information such as the number of votes, candidates, ballots or parties.



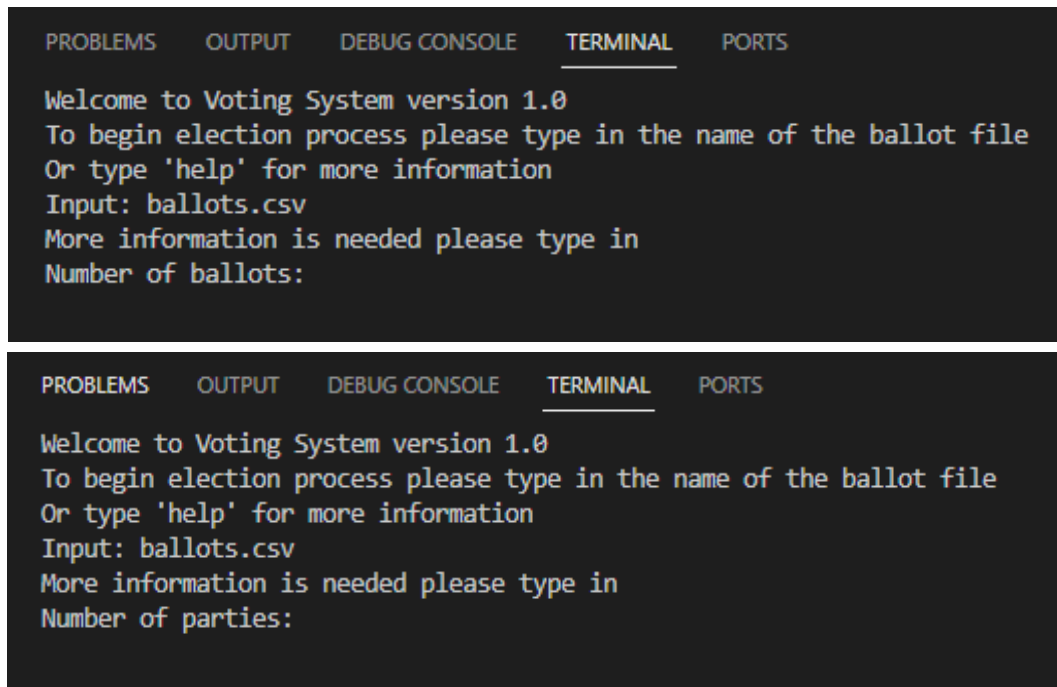
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: ballots.csv
More information is needed please type in
Number of votes:
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: ballots.csv
More information is needed please type in
Number of candidates:
```



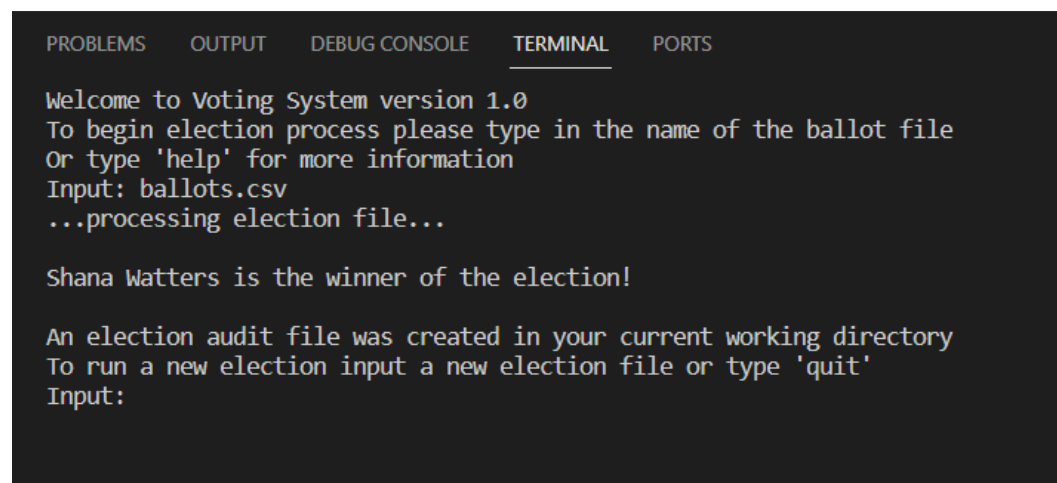
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: ballots.csv
More information is needed please type in
Number of ballots:

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: ballots.csv
More information is needed please type in
Number of parties:
```

When the user inputs a file, a message will be printed to them to let them know the system is working. It will then print the winner of the election and information about the audit file. It will then continue to ask for input until the user quits the system.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Welcome to Voting System version 1.0
To begin election process please type in the name of the ballot file
Or type 'help' for more information
Input: ballots.csv
...processing election file...

Shana Watters is the winner of the election!

An election audit file was created in your current working directory
To run a new election input a new election file or type 'quit'
Input:
```

### 6.3 Screen Objects and Actions

The voting system will be run and displayed through the command prompt so there are no screen objects associated with the actual display of the screen. Users can provide values for objects like votes, candidates, ballots, and parties when prompted as shown above.

## 7. REQUIREMENTS MATRIX

Use Case	System Component involved in given use case
UC_1 Prompting a User for file	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> NA
UC_2 Check if the file exists	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> NA
UC_3 Determine if the file is IR or CPL	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> electionType
UC_4 Save IR file Information	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> Candidates[] ; Party[] ; cBallot[] ; pName ; cName ; Party ; numBallots ; numSeats
UC_5 Save IR Ballots	<b>Class:</b> Election <b>Functions:</b> main() ; Ballot(int[] v, int numC, int numV) <b>Objects/Variables:</b> cBallot[] ; votes ; numCandidates ; numVotes
UC_6 Run IR Algorithm	<b>Class:</b> InstantRunoff <b>Functions:</b> IR(Ballot[] b, Candidate[] c) ; runElection(Ballot[] b) ; dropCandidate(Candidate[] c) ; redistributeVotes(Candidate[] c) ; coinFlip(Candidate[] c) <b>Objects/Variables:</b> ballots[] ; Candidates[] ; numberCandidates ; winner
UC_7 Name IR Audit File	<b>Class:</b> Election <b>Functions:</b> main()

	Objects/Variables: NA
UC_8 Produce IR Audit File	<p><b>Class:</b> AuditInfoCollector</p> <p><b>Functions:</b> getElecType() ; setElecType(String elec) ; getDate() ; setDate(String date) ; getCandidate() ; setCandidate(Candidate[] c) ; getParties() ; setParties(party[] p) ; getnBallots() ; setnBallots(ballots) ; getnSeats() ; setnSeats(0 ; getDCandidates() ; setDCandidates(Candidates[] dc) ; getWinners() ; setWinners(Candidate[] w)</p> <p><b>Objects/Variables:</b> electionType ; electionDate ; Candidate[] ; Party[] ; numBallots ; numSeats ; droppedCandidate[] ; winners[]</p>
UC_9 Save CPL file information	<p><b>Class:</b> Electron</p> <p><b>Functions:</b> main()</p> <p><b>Objects/Variables:</b> Candidates[] ; Party[] ; pBallot[] ; pName ; numBallots ; numSeats ; numPVotes</p>
UC_10 Save CPL Ballots	<p><b>Class:</b> Election</p> <p><b>Functions:</b> main()</p> <p><b>Objects/Variables:</b> Candidates[] ; Party[] ; pBallot[] ; pName ; cName ; Party ; numBallots ; numSeats ; numPVotes ; numCandidates ; numVotes</p>
UC_11 Run CPL Algorithm	<p><b>Class:</b> ClosedPartyList</p> <p><b>Functions:</b> CPL(Ballot[] b, Party[] p, int nSeats) ; runElection(Ballot[] b) ; assignSeats(Candidate[] c) ; coinFlip(Party[] p)</p> <p><b>Objects/Variables:</b> Ballots[] ; Party[] ; numParties ; numSeats ; winners[]</p>
UC_12 Name CPL Audit File	<p><b>Class:</b> Election</p> <p><b>Functions:</b> main()</p> <p><b>Objects/Variables:</b> NA</p>
UC_13 Produce CPL Audit File	<p><b>Class:</b> AuditInfoCollector</p> <p><b>Functions:</b> getElecType() ; setElecType(String elec) ; getDate() ; setDate(String date) ; getCandidate() ; setCandidate(Candidate[] c) ; getParties() ; setParties(party[] p) ; getnBallots() ; setnBallots(ballots) ; getnSeats() ; setnSeats(0 ; getDCandidates() ; setDCandidates(Candidates[] dc) ; getWinners() ; setWinners(Candidate[] w)</p>

	<b>Objects/Variables:</b> electionType ; electionDate ; Candidate[] ; Party[] ; numBallots ; numSeats ; droppedCandidate[] ; winners[]
UC_14 Coin Flip	<b>Class:</b> InstantRunoff ; ClosedPartyList <b>Functions:</b> coinFlip(Candidate[] c) ; coinFlip(Party[] p) <b>Objects/Variables:</b> Candidates[] ; Ballots [] ; Party[]
UC_15 Display Winners	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> winners[]
UC_16 Help Feature	<b>Class:</b> Election <b>Functions:</b> main() <b>Objects/Variables:</b> NA

## 8. APPENDICES

To be determined.