Voting System Software

Generated by Doxygen 1.9.6

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 countBallot Class Reference

Public Member Functions

- **countBallot** ()

3.1.1 Detailed Description

The **countBallot** (p. 5) program implements functions that are related to counting the ballots based on election type using the information given into the voting system software through a .csv file.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **16** of file **countBallot.java**.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 countBallot()

```
countBallot.countBallot ( )
```

A constructor of the **countBallot** (p. 5) class that takes in no parameter and will calculate the total number of ballots for each candidate.

Definition at line **28** of file **countBallot.java**.

The documentation for this class was generated from the following file:

- src/countBallot.java

3.2 countBallotTest Class Reference

Public Member Functions

- void **test1_countBallot** ()
- void **test2_countBallot** ()
- void **test3_countBallot** ()

3.2.1 Detailed Description

The **countBallot** (p. 5) program contains the test cases with different conditions to check whether the voting system software meets all its acceptance criteria by counting the ballots based on election type using the information given into the voting system software through a .csv file.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **18** of file **countBallotTest.java**.

3.2.2 Member Function Documentation

3.2.2.1 test1_countBallot()

```
void countBallotTest.test1_countBallot ( )
```

Definition at line **22** of file **countBallotTest.java**.

3.2.2.2 test2_countBallot()

```
void countBallotTest.test2_countBallot ( )
```

Definition at line **63** of file **countBallotTest.java**.

3.2.2.3 test3_countBallot()

```
void countBallotTest.test3_countBallot ( )
```

Definition at line **101** of file **countBallotTest.java**.

The documentation for this class was generated from the following file:

- src/countBallotTest.java

3.3 displayResults Class Reference

Public Member Functions

- **displayResults** ()
- void **generateAuditFile** ()
- void **showResults** ()

3.3.1 Detailed Description

The **displayResults** (p. 7) program implements functions that are related to displaying the final results of the election based on the election type as well as generate an audit file for a completed election.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **17** of file **displayResults.java**.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 displayResults()

```
displayResults.displayResults ( )
```

A constructor of the **displayResults** (p. 7) class that takes in no parameter and will display the results based on the final rankings.

Definition at line **23** of file **displayResults.java**.

3.3.3 Member Function Documentation

3.3.3.1 generateAuditFile()

```
void displayResults.generateAuditFile ( )
```

A function of the **displayResults** (p. 7) class that generates the audit file for completed elections based on election type.

Definition at line **32** of file **displayResults.java**.

3.3.3.2 showResults()

```
void displayResults.showResults ( )
```

A function of the **displayResults** (p. 7) class that shows the final result of the election based on the election type.

Definition at line **111** of file **displayResults.java**.

The documentation for this class was generated from the following file:

- src/displayResults.java

3.4 displayResultsTest Class Reference

Public Member Functions

- void **test1_generateAuditFile** ()
- void **test2_generateAuditFile** ()
- void **test3_showResults** ()
- void **test4_showResults** ()

3.4.1 Detailed Description

The **displayResults** (p. 7) program contains the test cases with different conditions to check whether the voting system software meets all its acceptance criteria by displaying the final results of the election based on the election type as well as generate an audit file for a completed election.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **19** of file **displayResultsTest.java**.

3.4.2 Member Function Documentation

3.4.2.1 test1_generateAuditFile()

```
void displayResultsTest.test1_generateAuditFile ( )
```

Definition at line **23** of file **displayResultsTest.java**.

3.4.2.2 test2_generateAuditFile()

```
void displayResultsTest.test2_generateAuditFile ( )
```

Definition at line **62** of file **displayResultsTest.java**.

3.4.2.3 test3_showResults()

```
void displayResultsTest.test3_showResults ( )
```

Definition at line **101** of file **displayResultsTest.java**.

3.4.2.4 test4_showResults()

```
void displayResultsTest.test4_showResults ( )
```

Definition at line **137** of file **displayResultsTest.java**.

The documentation for this class was generated from the following file:

- src/displayResultsTest.java

3.5 fileSystem Class Reference

Public Member Functions

- **fileSystem** ()

Static Public Member Functions

- static void **fileSystemRead** ()
- static void **openFile** (File fileName)
- static boolean **checkFileFormat** (File fileName)
- static void **readFile** (File fileName)
- static String **getFileExtension** (File fullName)

3.5.1 Detailed Description

The **fileSystem** (p. 10) program implements functions that are related to perform file handling within the voting system software.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **16** of file **fileSystem.java**.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 fileSystem()

```
fileSystem.fileSystem ( )
```

A constructor of the **fileSystem** (p. 10) class that takes in no parameter and will open the selected files. It displays a message indicating if the file can be opened.

Definition at line **34** of file **fileSystem.java**.

3.5.3 Member Function Documentation

3.5.3.1 checkFileFormat()

```
static boolean fileSystem.checkFileFormat (
             File fileName )  [static]
```

A function of the **fileSystem** (p. 10) class that reads in a file type and checks if the file format is correct.

Parameters

fileName	- a file type indicating the file name

Returns

boolean indicating if file format is correct

Definition at line **87** of file **fileSystem.java**.

3.5.3.2 fileSystemRead()

```
static void fileSystem.fileSystemRead ( )  [static]
```

Definition at line **44** of file **fileSystem.java**.

3.5.3.3 getFileExtension()

```
static String fileSystem.getFileExtension (
             File fullName )  [static]
```

A function that takes in a file type and gets the file extension

Parameters

fullName	- a file type indicating the name of the file

Returns

string indicating file type after "." in file name

Definition at line **242** of file **fileSystem.java**.

3.5.3.4 openFile()

```
static void fileSystem.openFile (
             File fileName )  [static]
```

A function of the **fileSystem** (p. 10) class that reads in a file type and opens the file if the file format is correct. It displays a message indicating if the file can be opened.

Parameters

fileName	- a file type indicating the file name

Definition at line **67** of file **fileSystem.java**.

3.5.3.5 readFile()

```
static void fileSystem.readFile (
             File fileName )  [static]
```

A function of the **fileSystem** (p. 10) class that takes in a file type and reads the CSV file starting from the first line to indicate election type.

Definition at line **105** of file **fileSystem.java**.

The documentation for this class was generated from the following file:

- src/fileSystem.java

3.6 fileSystemTest Class Reference

Public Member Functions

- void **test1_getFileExtension** ()
- void **test2_getFileExtension** ()
- void **test3_getFileExtension** ()
- void **test4_getFileExtension** ()
- void **test5_getFileExtension** ()
- void **test6_getFileExtension** ()
- void **test7_checkFileFormat** ()
- void **test8_checkFileFormat** ()
- void **test9_checkFileFormat** ()
- void **test10_checkFileFormat** ()
- void **test11_readFile** () throws IOException
- void **test12_readFile** () throws IOException
- void **test13_openFile** ()
- void **test14_openFile** ()

3.6.1 Detailed Description

The **fileSystemTest** (p. 13) program contains the test cases with different conditions to check whether the voting system software meets all its acceptance criteria by performing file handling in a correct manner.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **18** of file **fileSystemTest.java**.

3.6.2 Member Function Documentation

3.6.2.1 test10_checkFileFormat()

```
void fileSystemTest.test10_checkFileFormat ( )
```

Definition at line **104** of file **fileSystemTest.java**.

3.6.2.2 test11_readFile()

```
void fileSystemTest.test11_readFile ( ) throws IOException
```

Definition at line **113** of file **fileSystemTest.java**.

3.6.2.3 test12_readFile()

```
void fileSystemTest.test12_readFile ( ) throws IOException
```

Definition at line **144** of file **fileSystemTest.java**.

3.6.2.4 test13_openFile()

```
void fileSystemTest.test13_openFile ( )
```

Definition at line **165** of file **fileSystemTest.java**.

3.6.2.5 test14_openFile()

```
void fileSystemTest.test14_openFile ( )
```

Definition at line **183** of file **fileSystemTest.java**.

3.6.2.6 test1_getFileExtension()

```
void fileSystemTest.test1_getFileExtension ( )
```

Definition at line **23** of file **fileSystemTest.java**.

3.6.2.7 test2_getFileExtension()

```
void fileSystemTest.test2_getFileExtension ( )
```

Definition at line **32** of file **fileSystemTest.java**.

3.6.2.8 test3_getFileExtension()

```
void fileSystemTest.test3_getFileExtension ( )
```

Definition at line **41** of file **fileSystemTest.java**.

3.6.2.9 test4_getFileExtension()

```
void fileSystemTest.test4_getFileExtension ( )
```

Definition at line **50** of file **fileSystemTest.java**.

3.6.2.10 test5_getFileExtension()

```
void fileSystemTest.test5_getFileExtension ( )
```

Definition at line **59** of file **fileSystemTest.java**.

3.6.2.11 test6_getFileExtension()

```
void fileSystemTest.test6_getFileExtension ( )
```

Definition at line **68** of file **fileSystemTest.java**.

3.6.2.12 test7_checkFileFormat()

```
void fileSystemTest.test7_checkFileFormat ( )
```

Definition at line **77** of file **fileSystemTest.java**.

3.6.2.13 test8_checkFileFormat()

```
void fileSystemTest.test8_checkFileFormat ( )
```

Definition at line **86** of file **fileSystemTest.java**.

3.6.2.14 test9_checkFileFormat()

```
void fileSystemTest.test9_checkFileFormat ( )
```

Definition at line **95** of file **fileSystemTest.java**.

The documentation for this class was generated from the following file:

- src/fileSystemTest.java

3.7 finalRanking Class Reference

Public Member Functions

- **finalRanking** ()
- Map< String, List< Integer > > **checkForTie** ()
- int **fairCoinToss** ()
- int **poolCoinToss** (int range)

3.7.1 Detailed Description

The **finalRanking** (p. 16) program implements functions that are related to determine the final ranking of the candidates/parties of the election and perform check for tie and flip a coin toss if necessary to determine the winner through an unbiased method.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **17** of file **finalRanking.java**.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 finalRanking()

```
finalRanking.finalRanking ( )
```

A constructor of the **finalRanking** (p. 16) class that takes in no parameter and will checks if there is a tie.

Definition at line **25** of file **finalRanking.java**.

3.7.3 Member Function Documentation

3.7.3.1 checkForTie()

```
Map< String, List< Integer > > finalRanking.checkForTie ( )
```

A function that checks for tie and prompts users if they want to perform a coin toss. Performs a coin toss based on election type if there is a tie.

Returns

boolean indicating if there is a tie

Definition at line **34** of file **finalRanking.java**.

3.7.3.2 fairCoinToss()

```
int finalRanking.fairCoinToss ( )
```

A function that performs fair coin toss if there is a tie between two parties or candidates.

Returns

results

Definition at line **157** of file **finalRanking.java**.

3.7.3.3 poolCoinToss()

```
int finalRanking.poolCoinToss (
            int range )
```

A function that performs pool coin toss if there is a tie between parties or candidates based on the range.

Parameters

range	indicating the number of parties/ candidates with the same number of votes

Returns

integer indicating results

Definition at line **181** of file **finalRanking.java**.

The documentation for this class was generated from the following file:

- src/finalRanking.java

3.8 finalRankingTest Class Reference

Public Member Functions

- void **test1_checkForTie** ()
- void **test2_checkForTie** ()
- void **test3_checkForTie** ()
- void **test4_fairCoinToss** ()
- void **test5_poolCoinToss** ()

3.8.1 Detailed Description

The **finalRanking** (p. 16) program contains the test cases with different conditions to check whether the voting system software meets all its acceptance criteria by determining the final ranking of the candidates/parties of the election and perform check for tie and flip a coin toss if necessary to determine the winner through an unbiased method.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **19** of file **finalRankingTest.java**.

3.8.2 Member Function Documentation

3.8.2.1 test1_checkForTie()

```
void finalRankingTest.test1_checkForTie ( )
```

Definition at line **23** of file **finalRankingTest.java**.

3.8.2.2 test2_checkForTie()

```
void finalRankingTest.test2_checkForTie ( )
```

Definition at line **78** of file **finalRankingTest.java**.

3.8.2.3 test3_checkForTie()

```
void finalRankingTest.test3_checkForTie ( )
```

Definition at line **131** of file **finalRankingTest.java**.

3.8.2.4 test4_fairCoinToss()

```
void finalRankingTest.test4_fairCoinToss ( )
```

Definition at line **183** of file **finalRankingTest.java**.

3.8.2.5 test5_poolCoinToss()

```
void finalRankingTest.test5_poolCoinToss ( )
```

Definition at line **223** of file **finalRankingTest.java**.

The documentation for this class was generated from the following file:

- src/finalRankingTest.java

3.9 rankings Class Reference

Public Member Functions

- **rankings** ()
- Map< String, List< Integer > > **checkRanking** (Map< String, List< Integer > > ballotWithName)
- void **checkMajority** (Map< String, List< Integer > > ballotWithName)

3.9.1 Detailed Description

The rankings program implements functions that are related to determine the ranking of the candidates/parties of the election as well as check whether there is a majority to determine the winner of the election.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **18** of file **rankings.java**.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 rankings()

```
rankings.rankings ( )
```

A constructor of the rankings class that takes in no parameter and counts the ballot. The rankings are then determined.

Definition at line **30** of file **rankings.java**.

3.9.3 Member Function Documentation

3.9.3.1 checkMajority()

```
void rankings.checkMajority (
            Map< String, List< Integer > > ballotWithName )
```

A function that reads the totalBallots array and check if there is a majority among the candidates.

Parameters

totalBallots	— a 2D array indicating the number of votes per candidate

Definition at line **88** of file **rankings.java**.

3.9.3.2 checkRanking()

```
Map< String, List< Integer > > rankings.checkRanking (
            Map< String, List< Integer > > ballotWithName )
```

A function that reads the newBallots array and determines the ranking for each candidate based on number of votes

Parameters

newBallots	— a 2D array indicating the number of votes for each candidate
initialRank	- an integer indicating the initial rank of the candidate

Returns

integer array indicating candidates' rankings

Definition at line **42** of file **rankings.java**.

The documentation for this class was generated from the following file:

- src/rankings.java

3.10 rankingsTest Class Reference

Public Member Functions

- void **test1_checkRankingsTest** ()
- void **test2_checkRankingsTest** ()
- void **test3_checkRanking** ()
- void **test4_checkMajority** ()
- void **test5_checkMajority** ()

3.10.1 Detailed Description

The **rankingsTest** (p. 21) program contains the test cases with different conditions to check whether the voting system software meets all its acceptance criteria by determining the ranking of the candidates/parties of the election as well as check whether there is a majority to determine the winner of the election.

Author

Bryan Yen Sheng Lee

Cedric Tan Yee Shuen

Sherryl Ooi Shi Tyng

Version

2.0 @ since 2023-03-19

Definition at line **19** of file **rankingsTest.java**.

3.10.2 Member Function Documentation

3.10.2.1 test1_checkRankingsTest()

```
void rankingsTest.test1_checkRankingsTest ( )
```

Definition at line **24** of file **rankingsTest.java**.

3.10.2.2 test2_checkRankingsTest()

```
void rankingsTest.test2_checkRankingsTest ( )
```

Definition at line **56** of file **rankingsTest.java**.

3.10.2.3 test3_checkRanking()

```
void rankingsTest.test3_checkRanking ( )
```

Definition at line **86** of file **rankingsTest.java**.

3.10.2.4 test4_checkMajority()

```
void rankingsTest.test4_checkMajority ( )
```

Definition at line **117** of file **rankingsTest.java**.

3.10.2.5 test5_checkMajority()

```
void rankingsTest.test5_checkMajority ( )
```

Definition at line **151** of file **rankingsTest.java**.

The documentation for this class was generated from the following file:

- src/rankingsTest.java

3.11 votingSystem Class Reference

Static Public Member Functions

- static void **main** (String args[ ])

3.11.1 Detailed Description

Definition at line **1** of file **votingSystem.java**.

3.11.2 Member Function Documentation

3.11.2.1 main()

```
static void votingSystem.main (
             String args[ ] )  [static]
```

Definition at line **2** of file **votingSystem.java**.

The documentation for this class was generated from the following file:

- src/votingSystem.java

Chapter 4

File Documentation

4.1 countBallot.java

```java
import javax.swing.*;
import java.util.*;

public class countBallot {

    // An array list that stores the count of total ballots for each candidate
    static List<List<Integer» totalBallots = new ArrayList<>();

    // A HashMap that stores the count of total ballots for each candidate along with the candidate name
    static Map <String, List<Integer» ballotWithName = new HashMap<>();

    public countBallot () {

        // Create fileSystem object
        fileSystem files = new fileSystem();

        // If the election type is Closed Party Listing (CPL)
        if(fileSystem.electionType.equals("CPL")) {

            for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                List<Integer> subBallotCount = new ArrayList<>();
                subBallotCount.add(0);
                subBallotCount.add(0);
                totalBallots.add(subBallotCount);
            }

            // Loops through number of votes and candidates to allocate ballots based on parties
            for (int i = 0; i < fileSystem.numOfCandidates; i++) {
                for (int j = 0; j < fileSystem.numOfVotes; j++) {

                    // if the ballot received by the party is 1
                    if(fileSystem.ballot.get(j).get(i) == 1) {
                        int oldValue = totalBallots.get(i).get(1);
                        oldValue++; // increment the count by 1
                        totalBallots.get(i).set(1, oldValue); // update the number of 1s the party received
                    }

                    // if the ballot received by the party is 0
                    else {
                        int oldValue = totalBallots.get(i).get(0);
                        oldValue++; // increment the count by 1
                        totalBallots.get(i).set(0, oldValue); // update the total number of 0s the party received
                    }
                }
            }
        }

        // If the election type is Instant Runoff Voting (IR)
        else if(fileSystem.electionType.equals("IR")){

            for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                List<Integer> subBallotCount = new ArrayList<>();
                for(int j = 0; j <= fileSystem.numOfCandidates; j++) {
                    subBallotCount.add(0);
                }
                totalBallots.add(subBallotCount);
            }

            // Loops through number of votes and candidates to allocate the ballots based on the ranking of each candidate
            for (int i = 0; i < fileSystem.numOfCandidates; i++) {
                for (int j = 0; j < fileSystem.numOfVotes; j++) {
                    for (int k = 0; k <= fileSystem.numOfCandidates; k++) {
                        if(fileSystem.ballot.get(j).get(i) == k) {
                            int oldValue = totalBallots.get(i).get(k);
                            oldValue++; // increment the count by 1
                            totalBallots.get(i).set(k, oldValue); // update the total number of ballots the candidate received
                        }
                    }
                }
            }
        }

        // If the election type is neither IR nor CPL
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
        }

        for(int i = 0; i < fileSystem.numOfCandidates; i++) {
            ballotWithName.put(fileSystem.candidates.get(i), totalBallots.get(i));
        }

    }
}
```

4.2 countBallotTest.java

```java
import static org.junit.Assert.*;
import org.junit.Test;
import java.util.*;

public class countBallotTest {

    @Test
    // This test checks that countBallot() returns the correct TotalBallots list and ballotWithName map for CPL
    public void test1_countBallot(){

        fileSystem.electionType = "CPL";                                                   // set election type to IR
        fileSystem.numOfCandidates = 4;                                                    // set number of parties to 4
        fileSystem.numOfVotes = 7;                                                         // set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   // add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                     // add 7 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,0,1));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));

        // Create an object of the countBallot class to count totalBallots and ballotsWithName
        countBallot ballotCount = new countBallot();

        // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
        List<List<Integer» expectedTotalBallots = Arrays.asList(
            // (index 0: number of zeros in all ballots, index 1: number of ones in all ballots)
            Arrays.asList(5,2),
            Arrays.asList(4,3),
            Arrays.asList(6,1),
            Arrays.asList(6,1)
        );
        // Test that the totalBallots list has been calculated correctly using assertEquals
        assertEquals(expectedTotalBallots, ballotCount.totalBallots);

        // expectedBallotWithName is created and used to compare results with ballotCount.ballotWithName
        Map<String, List<Integer» expectedBallotWithName = new HashMap<>();
        expectedBallotWithName.put("Democratic", Arrays.asList(5,2));
        expectedBallotWithName.put("Republican", Arrays.asList(4,3));
        expectedBallotWithName.put("Reform", Arrays.asList(6,1));
        expectedBallotWithName.put("Green", Arrays.asList(6,1));
        // Test that the ballotWithName list has been calculated correctly using assertEquals
        assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
    }


    @Test
    // The second test checks that countBallot() returns the correct TotalBallots list and ballotWithName map for IR
    public void test2_countBallot(){

        fileSystem.electionType = "IR";                                                 // set election type to IR
        fileSystem.numOfCandidates = 3;                                                 // set number of candidates to 3
        fileSystem.numOfVotes = 4;                                                      // set number of votes to 4
        fileSystem.candidates.add("Sherryl Ooi");                                      // add first candidate
        fileSystem.candidates.add("Bryan Lee");                                        // add second candidate
        fileSystem.candidates.add("Cedric Tan");                                       // add third candidate
        fileSystem.ballot.add(Arrays.asList(1, 2, 3));                               // add 4 arbitrary ballots with rankings
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));
        fileSystem.ballot.add(Arrays.asList(1, 2, 3));
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));

        // Create an object of the countBallot class to count totalBallots and ballotsWithName
        countBallot ballotCount = new countBallot();

        // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
        List<List<Integer» expectedTotalBallots = Arrays.asList(
            // (index 0: number of zeros in all ballots, index 1: number of ones in all ballots)
            Arrays.asList(0, 3, 1, 0),
            Arrays.asList(0, 0, 2, 2),
            Arrays.asList(0, 1, 1, 2)
        );
        // Test that the totalBallots list has been calculated correctly using assertEquals
        assertEquals(expectedTotalBallots, ballotCount.totalBallots);

        // expectedBallotWithName is created and used to compare results with ballotCount.ballotWithName
        Map<String, List<Integer» expectedBallotWithName = new HashMap<>();
        expectedBallotWithName.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
        expectedBallotWithName.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
        expectedBallotWithName.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));
        // Test that the ballotWithName list has been calculated correctly using assertEquals
        assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
    }


    @Test
    // This test checks that countBallot() returns an empty TotalBallots list and ballotWithName map when election type is neither CPL or IR
    public void test3_countBallot(){

        fileSystem.electionType = "OPL";            // set election type to Open Party List

        // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
        List<List<Integer» expectedTotalBallots = Arrays.asList();
        countBallot ballotCount = new countBallot();
        // Invalid election type means that ballots will not be counted therefore the array list is empty
        assertEquals(expectedTotalBallots,ballotCount.totalBallots);

        // expectedBallotWithName is created and used to compare results with ballotCount.ballotWithName
        Map<String, List<Integer» expectedBallotWithName = new HashMap<>();
        // Invalid election type means that ballots with name will not be counted therefore the map is empty
        assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
    }

}


```

4.3 displayResults.java

```java
import java.io.*;
import javax.swing.*;
import java.util.*;
```

```java

public class displayResults {

    public displayResults () {
        finalRanking rank = new finalRanking();
        // showResults();
    }

    public void generateAuditFile () {
        try {

            // Create JFrame to display information
            JFrame parentFrame = new JFrame();
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setDialogTitle("Save audit file");

            int userSelection = fileChooser.showSaveDialog(parentFrame);

            // Checks whether approve (yes, ok) is chosen
            if (userSelection == JFileChooser.APPROVE_OPTION) {
                File outputFile = fileChooser.getSelectedFile();
                System.out.println("Save as file: " + outputFile.getAbsolutePath());

                // Generate output file
                outputFile.createNewFile();
                PrintWriter output = new PrintWriter(outputFile);
                StringBuffer csvData = new StringBuffer("");

                csvData.append("Election type: " + fileSystem.electionType + "\n");

                // If the election type is Closed Party Listing (CPL)
                if(fileSystem.electionType.equals("CPL")) {
                    // add election information into the audit file
                    csvData.append("Number of Parties: " + fileSystem.numOfCandidates + "\n");
                    csvData.append("Parties joined election: " + fileSystem.candidates + "\n");
                    for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                        csvData.append("Candidates of party " + fileSystem.candidates.get(i) + ": " +
fileSystem.candidatesList.get(i) + "\n");
                    }
                    csvData.append("Total seats elected: " + fileSystem.numOfSeats + "\n");
                    csvData.append("Total number of voters: " + fileSystem.numOfVotes + "\n");

                    // add ranking for each candidate into GUI
                    List<Map.Entry<String, List<Integer»> entries = new
ArrayList<>(rankings.ranking.entrySet());
                    List<String> keys = new ArrayList<>(rankings.ranking.keySet());
                    for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                        csvData.append(entries.get(i).getKey() + " is rank " +
(keys.indexOf(entries.get(i).getKey()) + 1) + ".\n");
                    }
                    csvData.append("The final winners of the election are parties: \n");

                    // add final winners’ ranking into the GUI
                    List<Map.Entry<String, List<Integer»> entries2 = new
ArrayList<>(finalRanking.finalRanking.entrySet());
                    List<String> keys2 = new ArrayList<>(finalRanking.finalRanking.keySet());
                    for(int i = 0; i < fileSystem.numOfSeats; i++) {
                        csvData.append(entries2.get(i).getKey() + " is rank " +
(keys2.indexOf(entries2.get(i).getKey()) + 1) + ".\n");
                    }
                }

                // If the election type is Instant Runoff Voting (IR)
                else if(fileSystem.electionType.equals("IR")){
                    // add election information into the audit file
                    csvData.append("Number of Candidates: " + fileSystem.numOfCandidates + "\n");
                    csvData.append("Candidates joined election: " + fileSystem.candidates + "\n");
                    csvData.append("Total number of voters: " + fileSystem.numOfVotes + "\n");

                    // add the final winner and displays the results below the final winner
                    List<Map.Entry<String, List<Integer»> entries = new
ArrayList<>(rankings.ranking.entrySet());
                    csvData.append("The final winner of the election is candidate "  +
entries.get(0).getKey() + "\n");
                    csvData.append("\nThe results are as below: \n");
                    for(int i = 0; i < rankings.displayList.size(); i++) {
                        csvData.append(rankings.displayList.get(i) + "\n");
                    }
                }

                output.write(csvData.toString());
                output.close();
            }
        }

        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void showResults () {
        // GUI interface to prompt user to choose whether to generate an audit file
        int yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to generate Audit File?","Generate Audit File",JOptionPane.YES_NO_OPTION);

        // If the user chose YES to generate an audit file
        if(yesOrNo == JOptionPane.YES_OPTION) {
            generateAuditFile();
        }

        // Create JFrame to display information
        JFrame frame = new JFrame ("Final Results");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 800);
        JTextArea textArea  = new JTextArea();

        // Adds the election type onto the first line of the JFrame
        textArea.append("Election type: " + fileSystem.electionType + "\n");

        // If the election type is Closed Party Listing (CPL)
        if(fileSystem.electionType.equals("CPL")) {
            // add election information into the GUI
            textArea.append("Number of Parties: " + fileSystem.numOfCandidates + "\n");
            textArea.append("Parties joined election: " + fileSystem.candidates + "\n");
            for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                textArea.append("Candidates of party " + fileSystem.candidates.get(i) + ": " + fileSystem.candidatesList.get(i) + "\n");
            }
            textArea.append("Total seats elected: " + fileSystem.numOfSeats + "\n");
            textArea.append("Total number of voters: " + fileSystem.numOfVotes + "\n");

            // add final winners into the GUI
            List<Map.Entry<String, List<Integer»> entries = new ArrayList<>(rankings.ranking.entrySet());
            List<String> keys = new ArrayList<>(rankings.ranking.keySet());
            for(int i = 0; i < fileSystem.numOfCandidates; i++) {
                textArea.append(entries.get(i).getKey() + " is rank " + (keys.indexOf(entries.get(i).getKey()) + 1) + ".\n");
            }
            textArea.append("The final winners of the election are parties: \n");

            // add final winners' ranking into the GUI
            List<Map.Entry<String, List<Integer»> entries2 = new ArrayList<>(finalRanking.finalRanking.entrySet());
            List<String> keys2 = new ArrayList<>(finalRanking.finalRanking.keySet());
            for(int i = 0; i < fileSystem.numOfSeats; i++) {
                textArea.append(entries2.get(i).getKey() + " is rank " + (keys2.indexOf(entries2.get(i).getKey()) + 1) + ".\n");
            }
        }

        // If the election type is Instant Runoff Voting (IR)
        else if(fileSystem.electionType.equals("IR")){
            // add election information into the GUI
            textArea.append("Number of Candidates: " + fileSystem.numOfCandidates + "\n");
            textArea.append("Candidates joined election: " + fileSystem.candidates + "\n");
            textArea.append("Total number of voters: " + fileSystem.numOfVotes + "\n");

            // add the final winner and displays the results below the final winner
            List<Map.Entry<String, List<Integer»> entries = new ArrayList<>(rankings.ranking.entrySet());
            textArea.append("The final winner of the election is candidate "  + entries.get(0).getKey() + "\n");
            textArea.append("\nThe results are as below: \n");
            for(int i = 0; i < rankings.displayList.size(); i++) {
                textArea.append(rankings.displayList.get(i) + "\n");
            }
        }

        // modifications to JFrame
        frame.add(textArea);
        textArea.setEditable(false);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

4.4 displayResultsTest.java

```
import org.junit.Test;
```

```java
import static org.junit.Assert.*;
import java.util.*;

public class displayResultsTest {

    @Test
    // This test case checks if generateAuditFile() is called to export an audit file when the election type is CPL
    public void test1_generateAuditFile(){

        fileSystem.electionType = "CPL";                                                             // set election type to IR
        fileSystem.numOfCandidates = 4;                                                              // set number of parties to 4
        fileSystem.numOfSeats = 5;                                                                   // set number of seats to 5
        fileSystem.numOfVotes = 5;                                                                   // set number of votes to 5
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   // add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                               // add 5 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));

        // Creates a new object of the displayResults class
        displayResults dR = new displayResults();

            // +--------------------+
            // | Results:           |
            // +--------------------+
            // | "Democratic",(3,2) |
            // | "Republican",(2,3) |
            // | "Reform",(5,0)     |
            // | "Green",  (5,0)    |
            // +--------------------+

        // Initialize boolean
        boolean CheckIfAuditFileIsGenerated = true;
        // Checks if the generateAuditFile() method is called using try and catch
        try {
            dR.generateAuditFile();
            dR.showResults();
        }
        catch (Exception e) { CheckIfAuditFileIsGenerated  = false; }
        assertFalse(CheckIfAuditFileIsGenerated);
    }


    @Test
    // This test case checks if generateAuditFile() is called to export an audit file when the election type is IR
    public void test2_generateAuditFile(){

        fileSystem.electionType = "IR";                                                    // set election type to IR
        fileSystem.numOfCandidates = 3;                                                    // set number of candidates to 3
        fileSystem.numOfVotes = 5;                                                         // set number of votes to 5
        fileSystem.candidates.add("Cedric Tan");                                         // add first candidate
        fileSystem.candidates.add("Bryan Lee");                                          // add second candidate
        fileSystem.candidates.add("Sherryl Ooi");                                        // add third candidate
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));                                 // add 5 arbitrary ballots with rankings
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));

        // Creates a new object of the displayResults class
        displayResults dR = new displayResults();

            // +------------------------------+
            // | Results:                     |
            // +------------------------------+
            // | "Cedric",(0,0,2,3)           |
            // | "Bryan",(0,1,3,1)            |
            // | "Sherryl",(0,4,0,1)          |
            // +------------------------------+

        // Initialize boolean
        boolean CheckIfAuditFileIsGenerated = true;
        // Checks if the generateAuditFile() method is called using try and catch
        try {
            dR.generateAuditFile();
            dR.showResults();
        }
        catch (Exception e) { CheckIfAuditFileIsGenerated  = false; }
        assertTrue(CheckIfAuditFileIsGenerated);
    }


    @Test
    // This test case checks if showResults() is called to display the election results when the
election type is IR
    public void test3_showResults(){

        fileSystem.electionType = "CPL";
// set election type to IR
        fileSystem.numOfCandidates = 4;
// set number of parties to 4
        fileSystem.numOfSeats = 6;
// set number of seats to 4
        fileSystem.numOfVotes = 5;
// set number of votes to 5
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   //
add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                             //
add 5 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));

        // Creates a new object of the displayResults class
        displayResults dR = new displayResults();

            // +--------------------+
            // | Results:           |
            // +--------------------+
            // | "Democratic",(3,2) |
            // | "Republican",(2,3) |
            // | "Reform",(5,0)     |
            // | "Green",  (5,0)    |
            // +--------------------+

        // Initialize boolean
        boolean CheckIfResultsAreDisplayed = true;
        // Checks if the showResults() method is called using try and catch
        try {  dR.showResults();
               dR.generateAuditFile();} catch (Exception e) { CheckIfResultsAreDisplayed  = false; }
        assertFalse(CheckIfResultsAreDisplayed);
    }


    @Test
    // This test case checks if showResults() is called to display the election results when the
election type is IR
    public void test4_showResults(){

        fileSystem.electionType = "IR";                                                  // set
election type to IR
        fileSystem.numOfCandidates = 3;                                                  // set number
of candidates to 3
        fileSystem.numOfVotes = 5;                                                       // set number
of votes to 5
        fileSystem.candidates.add("Cedric Tan");                                       // add first
candidate
        fileSystem.candidates.add("Bryan Lee");                                        // add second
candidate
        fileSystem.candidates.add("Sherryl Ooi");                                      // add third
candidate
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));                              // add 5 arbitrary
ballots with rankings
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));

        // Creates a new object of the displayResults class
        displayResults dR = new displayResults();

            // +-----------------------------+
            // | Results:                    |
            // +-----------------------------+
            // | "Cedric",(0,0,2,3)          |
            // | "Bryan",(0,1,3,1)           |
            // | "Sherryl",(0,4,0,1)         |
            // +-----------------------------+

        // Initialize boolean
        boolean CheckIfResultsAreDisplayed= true;
        // Checks if the showResults() method is called using try and catch
        try {  dR.showResults(); } catch (Exception e) { CheckIfResultsAreDisplayed  = false; }
        assertTrue(CheckIfResultsAreDisplayed);
    }

}

```

4.5 fileSystem.java

```java
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.JFileChooser;

public class fileSystem {

    // Instance variables that are used in fileSystem class
    static String electionType;
    static int numOfCandidates;
    static List<String> candidates = new ArrayList<>();
    static int numOfSeats;
    static int numOfVotes;
    static List<List<Integer>> ballot = new ArrayList<>();
    static List<List<String>> candidatesList = new ArrayList<>();
    static int fileCount = 0;


    public fileSystem() {
        fileSystemRead();
        int yesOrNo = JOptionPane.showConfirmDialog(null,"Do you have other ballot files","Multiple file",JOptionPane.YES_NO_OPTION);
        while(yesOrNo == JOptionPane.YES_OPTION) {
            fileSystemRead();
            yesOrNo = JOptionPane.showConfirmDialog(null,"Do you have other ballot files","Multiple file",JOptionPane.YES_NO_OPTION);
        }
        JOptionPane.showMessageDialog(null,"You have a total of " + fileCount + " files opened.","INFO MESSAGE",JOptionPane.INFORMATION_MESSAGE);
    }

    public static void fileSystemRead() {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showOpenDialog(null);

        // Checks selected file to see if approve (yes, ok) is chosen
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();

            // Opens selected file
            openFile(selectedFile);
        }

        else {
            JOptionPane.showMessageDialog(null,"There is an error opening the file.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void openFile(File fileName) {

        // checks if file format is correct
        if(checkFileFormat(fileName)) {
            JOptionPane.showMessageDialog(null,"The file is open.","INFO MESSAGE",JOptionPane.INFORMATION_MESSAGE);
            readFile(fileName);
        }

        // Displays error message if an error is detected
        else {
            JOptionPane.showMessageDialog(null,"There is an error opening the file.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static boolean checkFileFormat (File fileName) {

        // checks if file type is a .csv file type
        if(getFileExtension(fileName).equals("csv")) {
            return true;
        }

        if (fileName == null) {
            return false;
        }

        return false;
    }

    public static void readFile (File fileName) {

        // Read lines inside CSV file
        try {

            // Variables to store information that is read from CSV file
            if(fileCount == 0) {
                // They read the first file
                BufferedReader file = new BufferedReader(new FileReader(fileName));
                electionType = file.readLine();
                numOfCandidates = Integer.parseInt(file.readLine());
                // allCandidates = file.readLine();
                // String[] subCandidates = allCandidates.split(", ");
                candidates.addAll(Arrays.asList(file.readLine().split(", ")));

                // Read in values based on election type
                if(electionType.equals("CPL")) {
                    for(int i = 0; i < numOfCandidates; i++) {
                        // List<String> subCandidatesList = new ArrayList<>(file.readLine());
                        candidatesList.add(i, Arrays.asList(file.readLine()));
                    }
                    numOfSeats = Integer.parseInt(file.readLine());
                    numOfVotes = Integer.parseInt(file.readLine());
                }

                else if (electionType.equals("IR")) {
                    numOfVotes = Integer.parseInt(file.readLine());
                }

                // Invalid election type
                else {
                    JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
                }

                String CurrentLine;

                // While loop to read in number of candidates from CSV file
                while ((CurrentLine = file.readLine()) != null)
                {
                    // Splits the line into array of strings based on commas
                    String [] fileLine = CurrentLine.split(",", -1);
                    List<Integer> subBallot = new ArrayList<>();
                    // Adds the candidates' ranking into ballot array
                    for(int i = 0; i < numOfCandidates; i++) {
                        if(fileLine[i] != "") {
                            subBallot.add(Integer.parseInt(fileLine[i]));
                        }
                        else {
                            subBallot.add(0);
                        }
                    }
                    ballot.add(subBallot);
                }

                // Close file
                file.close();

                // Update the count variable for the number of files opened
                fileCount ++;
            }
            else if(fileCount > 0) {
                // Read other files
                BufferedReader file = new BufferedReader(new FileReader(fileName));

                // readLine of new file to check if the header is identical
                String electionTypeNew = file.readLine();
                int numOfCandidatesNew = Integer.parseInt(file.readLine());
                List<String> candidatesNew = new ArrayList<>();
                candidatesNew.addAll(Arrays.asList(file.readLine().split(", ")));

                if(electionTypeNew.equals(electionType) && numOfCandidates == numOfCandidatesNew &&
candidatesNew.equals(candidates))
                {
                    // Read in values based on election type
                    // If the election type is Closed Party Listing (CPL)
                    if(electionType.equals("CPL")) {
                        List<List<String» candidatesListNew = new ArrayList<>();
                        for(int i = 0; i < numOfCandidates; i++) {
                            candidatesListNew.add(i, Arrays.asList(file.readLine()));
                        }
                        int numOfSeatsNew = Integer.parseInt(file.readLine());
                        int numOfVotesNew = Integer.parseInt(file.readLine());
                    }

                    // If the election type is Instant Runoff Voting (IR)
                    else if (electionType.equals("IR")) {
                        int numOfVotesNew = Integer.parseInt(file.readLine());
                    }

                    // If the election type is invalid
                    else {
                        JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
                    }

                    String CurrentLine;

                    // While loop to read in number of candidates from CSV file
                    while ((CurrentLine = file.readLine()) != null)
                    {
                        // Splits the line into array of strings based on commas
                        String [] fileLine = CurrentLine.split(",", -1);
                        List<Integer> subBallot = new ArrayList<>();
                        // Adds the candidates’ ranking into ballot array
                        for(int i = 0; i < numOfCandidates; i++) {
                            if(fileLine[i] != "") {
                                subBallot.add(Integer.parseInt(fileLine[i]));
                            }
                            else {
                                subBallot.add(0);
                            }
                        }
                        ballot.add(subBallot);
                    }
                    // Close file
                    file.close();

                    // Update the count variable for the number of files opened
                    fileCount++;
                }
                else {
                    JOptionPane.showMessageDialog(null,"The file does not have the same header.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
                }
            }
        }

        // Displays error message if an error is detected
        catch (Exception e) {
            JOptionPane.showMessageDialog(null,"There is an error opening the file.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }

    }

    public static String getFileExtension(File fullName) {

        String fileName = fullName.getName();
        int dotIndex = fileName.lastIndexOf(’.’);

        // Returns string indicating file type after "." in file name
        return (dotIndex == -1) ? "" : fileName.substring(dotIndex + 1);
    }
}
```

4.6 fileSystemTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;
import java.io.*;
import java.util.*;

public class fileSystemTest {


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test1_getFileExtension(){
        File testFile1 = new File("testfile.csv");
        // Expects "csv" as the outcome
        assertEquals("csv", fileSystem.getFileExtension(testFile1));
    }


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test2_getFileExtension(){
        File testFile2 = new File("testfile.pdf");
        // Expects "pdf" as the outcome
        assertEquals("pdf", fileSystem.getFileExtension(testFile2));
    }


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test3_getFileExtension(){
        File testFile3 = new File("testfiledocs");
        // Expects "" as the outcome
        assertEquals("", fileSystem.getFileExtension(testFile3));
    }


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test4_getFileExtension(){
        File testFile4 = new File("");
        // Expects "" as the outcome
        assertEquals("", fileSystem.getFileExtension(testFile4));
    }


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test5_getFileExtension(){
        File testFile5 = new File("testfile.csv.csv");
        // Expects "csv.csv" as the outcome
        assertEquals("csv", fileSystem.getFileExtension(testFile5));
    }


    @Test
    // This test checks that getFileExtension() returns the correct file extension
    public void test6_getFileExtension(){
        File testFile6 = new File("testfile..");
        // Expects "" as the outcome
        assertEquals("", fileSystem.getFileExtension(testFile6));
    }


    @Test
    // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
    public void test7_checkFileFormat(){
        File testFile7 = new File("testfile.csv");
        // Expects True as the outcome
        assertEquals(true, fileSystem.checkFileFormat(testFile7));
    }


    @Test
    // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
    public void test8_checkFileFormat(){
        File testFile8 = new File("testfile.pdf");
        // Expects False as the outcome
        assertEquals(false, fileSystem.checkFileFormat(testFile8));
    }


    @Test
    // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
    public void test9_checkFileFormat(){
        File testFile9 = new File("testfile..");
        // Expects False as the outcome
        assertEquals(false, fileSystem.checkFileFormat(testFile9));
    }


    @Test
    // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
    public void test10_checkFileFormat(){
        File testFile10 = new File("testfile.csv.csv");
        // Expects True as the outcome
        assertEquals(true, fileSystem.checkFileFormat(testFile10));
    }


    @Test
    // This test checks that readFile() reads and returns correct information based on election file
    public void test11_readFile() throws IOException{

        // Create temporary csv fie containing election information
        File testFile11 = File.createTempFile("testFile11", ".csv");
        FileWriter writer = new FileWriter(testFile11);
        writer.write("CPL\n3\nDemocratic, Republican, New Wave\nFoster, Volz, Pike\nGreen, Xu,
Wang\nJacks, Rosen\n3\n8");
        writer.close();

        // Call the method being tested
        fileSystem.readFile(testFile11);

        // Stores candidates from all parties
        List<List<String» expectedCandidatesList = Arrays.asList(
            // List of candidates from each party
            Arrays.asList("Foster, Volz, Pike"),
            Arrays.asList("Green, Xu, Wang"),
            Arrays.asList("Jacks, Rosen")
        );

        // Test that election information were read correctly from testFile12
        assertEquals("CPL", fileSystem.electionType);
        assertEquals(3, fileSystem.numOfCandidates);
        assertEquals(Arrays.asList("Democratic", "Republican", "New Wave"), fileSystem.candidates);
        assertEquals(expectedCandidatesList, fileSystem.candidatesList);
        assertEquals(3, fileSystem.numOfSeats);
        assertEquals(8, fileSystem.numOfVotes);
    }


    @Test
    // This test checks that readFile() reads and returns correct information based on election file
    public void test12_readFile() throws IOException{

        // Create temporary csv fie containing election information
        File testFile12 = File.createTempFile("testFile12", ".csv");
        FileWriter writer = new FileWriter(testFile12);
        writer.write("IR\n4\nRosen (D), Kleinberg (R), Chou (I), Royce (L)\n8");
        writer.close();

        // Call the method being tested
        fileSystem.readFile(testFile12);

        // Test that election information were read correctly from testFile12
        assertEquals("IR", fileSystem.electionType);
        assertEquals(4, fileSystem.numOfCandidates);
        assertEquals(Arrays.asList("Rosen (D)", "Kleinberg (R)", "Chou (I)", "Royce (L)"),
fileSystem.candidates);
        assertEquals(8, fileSystem.numOfVotes);
    }


    @Test
    // This test checks that openFile() checks the file format and then reads file
    public void test13_openFile(){
        File testFile13 = new File("Project2/testing/CPL_18-3-2023.csv");

        // Check if file format is True
        fileSystem.openFile(testFile13);
        assertTrue(fileSystem.checkFileFormat(testFile13));

        // Initialize boolean
        boolean functionWasRun = true;
        // Check if readFile() is called using try and catch
        try { fileSystem.readFile(testFile13); } catch (Exception e) { functionWasRun = true; }
        assertTrue(functionWasRun);

    }


    @Test
    // This test checks that openFile() checks the file format and then reads file
    public void test14_openFile(){
        File testFile14 = new File("Project1/testing/CPL_18-3-2023.pdf");

        fileSystem.openFile(testFile14);
        assertFalse(fileSystem.checkFileFormat(testFile14));

        // Check if ReadFile() function was run
        boolean functionWasRun = false;
        try { fileSystem.readFile(testFile14); } catch (Exception e) { functionWasRun = false; }
        assertFalse(functionWasRun);
    }

}




```

4.7 finalRanking.java

```
import javax.swing.*;
import java.lang.Math;
import java.util.*;

public class finalRanking {

    static Map<String, List<Integer» finalRanking = new LinkedHashMap<>();

    public finalRanking () {
        finalRanking = checkForTie();
    }

    public Map<String, List<Integer» checkForTie () {
        rankings rank = new rankings();
        int yesOrNo;
        int finalResult;

        // If the election type is Closed Party Listing (CPL)
        if(fileSystem.electionType.equals("CPL")) {
            List<Map.Entry<String, List<Integer»> entries = new ArrayList<>(rankings.ranking.entrySet());
            int totalSeats = fileSystem.numOfSeats;
            // check if the last rank from num of seats is tie with the next one
            for(int i = totalSeats - 1; i < fileSystem.numOfCandidates; i++) {
                if(entries.get(i).getValue().get(1) == entries.get(i+1).getValue().get(1)) {
                    if(entries.get(i+1).getValue().get(1) == entries.get(i+2).getValue().get(1)){
                        // if more than 2 have equal value
                        yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a pool coin toss","Pool Coin Toss",JOptionPane.YES_NO_OPTION);
                        if(yesOrNo == JOptionPane.YES_OPTION) {
                            // if yes, run a pool coin toss
                            finalResult = poolCoinToss(i + 2);
                            if(finalResult == 0) {
                                for(int k = 0; k < totalSeats - 1; k++) {
                                    finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
                                }
                                finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
                            }
                            else if(finalResult == 1){
                                for(int k = 0; k < totalSeats - 1; k++) {
                                    finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
                                }
                                finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
                            }
                            else {
                                for(int k = 0; k < totalSeats - 1; k++) {
                                    finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
                                }
                                finalRanking.put(entries.get(i+2).getKey(), entries.get(i).getValue());
                            }
                            return finalRanking;
                        }
                    }
                    else {
                        // if only 2 has equal value
                        yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a fair coin toss","Fair Coin Toss",JOptionPane.YES_NO_OPTION);
                        if(yesOrNo == JOptionPane.YES_OPTION) {
                            // if yes, run a fair coin toss.
                            finalResult = fairCoinToss();
                            if(finalResult == 0){
                                for(int k = 0; k < totalSeats - 1; k++) {
                                    finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
                                }
                                finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
                            }
                            else {
                                for(int k = 0; k < totalSeats - 1; k++) {
                                    finalRanking.put(entries.get(k).getKey(), entries.get(i+1).getValue());
                                }
                                finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
                            }
                            return finalRanking;                        }
                    }
                }
                else {
                    for(int k = 0; k < totalSeats; k++) {
                        finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
                    }
                }
            }
            return finalRanking;
        }

        // If the election type is Instant Runoff Voting (IR)
        else if(fileSystem.electionType.equals("IR")) {
            List<Map.Entry<String, List<Integer»> entries = new ArrayList<>(rankings.ranking.entrySet());
            int i = 0;
            // check if there is a tie between the first place
            if(entries.get(i).getValue().get(1) == entries.get(i+1).getValue().get(1)) {
                if(entries.get(i+1).getValue().get(1) == entries.get(i+2).getValue().get(1)){
                    // check if more than 2 has tie
                    yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a pool coin toss","Pool Coin Toss",JOptionPane.YES_NO_OPTION);
                    if(yesOrNo == JOptionPane.YES_OPTION) {
                        // run a pool coin toss if there is a tie
                        finalResult = poolCoinToss(i + 2);
                        if(finalResult == 0) {
                            finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
                        }
                        else if(finalResult == 1){
                            finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
                        }
                        else {
                            finalRanking.put(entries.get(i+2).getKey(), entries.get(i).getValue());
                        }
                    }
                }
                else {
                    // if there is only 2 has tie
                    yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a fair coin toss","Fair Coin Toss",JOptionPane.YES_NO_OPTION);
                    if(yesOrNo == JOptionPane.YES_OPTION) {
                        // run a fair coin toss if there is a tie
                        finalResult = fairCoinToss();
                        if(finalResult == 0) {
                            finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
                        }
                        else {
                            finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
                        }
                    }
                }
            }
            return finalRanking;
        }

        // If the election type is invalid
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
        }

        return finalRanking;
    }

    public int fairCoinToss () {
        // If the election type is Closed Party Listing (CPL)
        if(fileSystem.electionType.equals("CPL")) {
            return (int)(Math.random() * 2) + 0;
        }

        // If the election type is Instant Runoff Voting (IR)
        else if(fileSystem.electionType.equals("IR")) {
            return (int)(Math.random() * 2) + 0;
        }

        // Displays error message if an error is detected
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
        return 0;
    }

    public int poolCoinToss (int range) {
        // If the election type is Closed Party Listing (CPL)
        if(fileSystem.electionType.equals("CPL")) {
            return (int)(Math.random() * range) + 0;
        }

        // If the election type is Instant Runoff Voting (IR)
        else if(fileSystem.electionType.equals("IR")) {
            return (int)(Math.random() * range) + 0;
        }

        // Displays error message if an error is detected
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
        return 0;
    }
}
```

4.8 finalRankingTest.java

```
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.*;

public class finalRankingTest {

    @Test
    // This test case checks if checkfortie() performs pool coin toss when election type is CPL
    public void test1_checkForTie(){

        fileSystem.electionType = "CPL";
// set election type to IR
        fileSystem.numOfCandidates = 4;
// set number of parties to 4
        fileSystem.numOfSeats = 5;
// set number of seats to 5
        fileSystem.numOfVotes = 9;
// set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   //
add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                              //
add 9 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));

        // create an object of the rankings class and count ballots for each candidate
        rankings rank = new rankings();
        rank.checkRanking(countBallot.ballotWithName);

            // +--------------------+
            // | Results:           |
            // +--------------------+
            // | "Democratic",(6,3) |
            // | "Republican",(6,3) |
            // | "Reform",(6,3)     |
            // | "Green",  (9,0)    |
            // +--------------------+

        // create an object of the finalrankings class to check for tie
        finalRanking fr = new finalRanking();

        // Check if checkForTie() function was executed
        boolean CheckForTieWasRun = true;
        try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun  = false; }
        assertTrue(CheckForTieWasRun );

        // Check if poolCoinToss() was run when more than two parties have equal votes
        boolean poolCoinTossWasRun = true;
        try { fr.poolCoinToss(3); } catch (Exception e) {  poolCoinTossWasRun = false; }
        // poolCoinToss was run because a tie exist between more than two parties
        assertTrue(poolCoinTossWasRun);

        // Check if fairCoinToss() was not run given that poolCoinToss() was run
        boolean fairCoinTossWasRun = false;
        try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = true; }
        // fairCoinToss was not run because a tie exist between more than two parties
        assertFalse(fairCoinTossWasRun);

    }


    @Test
    // This test case checks if checkfortie() performs fair coin toss when election type is IR
    public void test2_checkForTie(){

        fileSystem.electionType = "IR";                                                  // set election type to IR
        fileSystem.numOfCandidates = 3;                                                  // set number of candidates to 3
        fileSystem.numOfVotes = 8;                                                       // set number of votes to 4
        fileSystem.candidates.add("Cedric Tan");                                        // add first candidate
        fileSystem.candidates.add("Bryan Lee");                                         // add second candidate
        fileSystem.candidates.add("Sherryl Ooi");                                       // add third candidate
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));                                 // add 4 arbitrary ballots with rankings
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));

        // create an object of the rankings class and count ballots for each candidate
        rankings rank = new rankings();
        rank.checkRanking(countBallot.ballotWithName);

            // +------------------------------+
            // | Results:                     |
            // +------------------------------+
            // | "Cedric",(0,0,2,6)           |
            // | "Bryan",(0,4,3,1)            |
            // | "Sherryl",(0,4,3,1)          |
            // +------------------------------+

        // create an object of the finalrankings class to check for tie
        finalRanking fr = new finalRanking();

        // Check if checkForTie() function was executed
        boolean CheckForTieWasRun = true;
        try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun  = false; }
        assertTrue(CheckForTieWasRun);

        // Check if poolCoinToss() was not run given that there is only a tie between two parties
        boolean poolCoinTossWasRun = false;
        try { fr.poolCoinToss(3); } catch (Exception e) {  poolCoinTossWasRun = true; }
        // poolCoinToss was run because a tie exist between two parties
        assertFalse(poolCoinTossWasRun);

        // Check if fairCoinToss() was run given that there is only a tie between two parties
        boolean fairCoinTossWasRun = true;
        try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = false; }
        // fairCoinToss was run because a tie exist between two parties
        assertTrue(fairCoinTossWasRun);
    }


    @Test
    // This test case checks if checkfortie() does not perform any toss given that there is no tie when election type is CPL
    public void test3_checkForTie(){

        fileSystem.electionType = "CPL";                                                                                                                    // set election type to IR
        fileSystem.numOfCandidates = 4;                                                                                                                    // set number of parties to 4
        fileSystem.numOfSeats = 5;                                                                                                                    // set number of seats to 5
        fileSystem.numOfVotes = 9;                                                                                                                    // set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   // add parties
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));                                              //
```

```
add 9 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));

        // create an object of the rankings class and count ballots for each candidate
        rankings rank = new rankings();
        rank.checkRanking(countBallot.ballotWithName);

            // +--------------------+
            // | Results:           |
            // +--------------------+
            // | "Democratic",(5,4) |
            // | "Republican",(7,2) |
            // | "Reform",(6,3)     |
            // | "Green",  (9,0)    |
            // +--------------------+

        // create an object of the finalrankings class to check for tie
        finalRanking fr = new finalRanking();

        // Test that checkForTie() function was executed
        boolean CheckForTieWasRun = true;
        try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun  = false; }
        assertTrue(CheckForTieWasRun );

        // Test that poolCoinToss() was not run given there is no tie
        boolean poolCoinTossWasRun = false;
        try { fr.poolCoinToss(3); } catch (Exception e) {  poolCoinTossWasRun = true; }
        assertFalse(poolCoinTossWasRun);

        // Test that fairCoinToss() was not run given there is no tie
        boolean fairCoinTossWasRun = false;
        try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = true; }
        assertFalse(fairCoinTossWasRun);
    }


    @Test
    // This test case checks if checkfortie() does not perform any toss if election type is CPL
    public void test4_fairCoinToss(){

        fileSystem.electionType = "IR";                                                 // set
election type to IR
        fileSystem.numOfCandidates = 3;                                                 // set number
of candidates to 3
        fileSystem.numOfVotes = 8;                                                      // set number
of votes to 4
        fileSystem.candidates.add("Cedric Tan");                                        // add first
candidate
        fileSystem.candidates.add("Bryan Lee");                                         // add second
candidate
        fileSystem.candidates.add("Sherryl Ooi");                                       // add third
candidate
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));                               // add 4 arbitrary
ballots with rankings
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));
        fileSystem.ballot.add(Arrays.asList(3, 1, 2));

        // create an object of the rankings class and count ballots for each candidate
        rankings rank = new rankings();
        rank.checkRanking(countBallot.ballotWithName);

            // +------------------------------+
            // | Results:                     |
            // +------------------------------+
            // | "Cedric",(0,0,2,6)           |
            // | "Bryan",(0,4,3,1)            |
            // | "Sherryl",(0,4,3,1)          |
            // +------------------------------+

        // create an object of the finalrankings class to check for tie
        finalRanking fr = new finalRanking();

        // Test that fairCoinToss returns a value between 0 and 1 indicating the winning party
        int result = fr.fairCoinToss();
        assertTrue(result >= 0 && result <= 1);
```

```java

    }

    @Test
    // This test case checks if checkfortie() does not perform any toss if election type is CPL
    public void test5_poolCoinToss(){

        fileSystem.electionType = "CPL";                    // set election type to IR
        fileSystem.numOfCandidates = 4;                     // set number of parties to 4
        fileSystem.numOfSeats = 5;                          // set number of seats to 5
        fileSystem.numOfVotes = 9;                          // set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   // add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                             // add 9 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));

        // create an object of the rankings class and count ballots for each candidate
        rankings rank = new rankings();
        rank.checkRanking(countBallot.ballotWithName);

            // +--------------------+
            // | Results:           |
            // +--------------------+
            // | "Democratic",(6,3) |
            // | "Republican",(6,3) |
            // | "Reform",(6,3)     |
            // | "Green",  (9,0)    |
            // +--------------------+

        // create an object of the finalrankings class to check for tie
        finalRanking fr = new finalRanking();

        // Test that fairCoinToss returns a value between 0 and 1 indicating the winning party
        int result = fr.poolCoinToss(3);
        assertTrue(result >= 0 && result <= 2);
    }



}
```

4.9 rankings.java

```java
import javax.swing.*;
import java.lang.Math;
import java.util.*;
import java.util.Map.*;

public class rankings {

    // A Linked Hash Map to store rankings of each candidate
    static Map<String, List<Integer>> ranking = new LinkedHashMap<>();

    // An array list to store ??
    static List<Map<String, List<Integer>>> displayList = new ArrayList<>();

    public rankings () {
        countBallot ballotCount = new countBallot();
        checkMajority(ballotCount.ballotWithName);
    }

    public Map<String, List<Integer>> checkRanking (Map<String, List<Integer>> ballotWithName) {

        // Check if election type is CPL
        if(fileSystem.electionType.equals("CPL")) {
            List<Map.Entry<String, List<Integer>>> entryList = new ArrayList<>(ballotWithName.entrySet());

            // Sort the entryList in descending order by second value in each list
            Collections.sort(entryList, Comparator.comparingInt((Map.Entry<String, List<Integer>> e) -> e.getValue().get(1)).reversed());

            // Create a new LinkedHashMap with the sorted entries
            for (Map.Entry<String, List<Integer» entry : entryList) {
                ranking.put(entry.getKey(), entry.getValue());
            }

            // Print the sorted map
            return ranking;
        }

        // Check if election type is IR
        else if (fileSystem.electionType.equals("IR")) {
            List<Map.Entry<String, List<Integer»> entryList = new ArrayList<>(ballotWithName.entrySet());

            // Sort the entryList in descending order by second value in each list
            Collections.sort(entryList, Comparator.comparingInt((Map.Entry<String, List<Integer» e) -> e.getValue().get(1)).reversed());

            // Create a new LinkedHashMap with the sorted entries
            for (Map.Entry<String, List<Integer» entry : entryList) {
                ranking.put(entry.getKey(), entry.getValue());
            }

            // Print the sorted map
            return ranking;
        }

        // Displays error message if an error is detected
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
        return ranking;
    }

    public void checkMajority (Map<String, List<Integer» ballotWithName) {

        // Checks if election type is CPL
        if(fileSystem.electionType.equals("CPL")) {
            ranking = checkRanking(ballotWithName);
        }

        // Checks if election type is IR
        else if (fileSystem.electionType.equals("IR")) {
            int majority = (int) fileSystem.numOfVotes * 50 / 100;
            ranking = checkRanking(ballotWithName);
            Map.Entry<String, List<Integer»entry = ranking.entrySet().iterator().next();
            int initialRank = 1;

            // While there is no majority
            while(entry.getValue().get(1) < majority) {

                displayList.add(new LinkedHashMap<>(ranking));
                List<Entry<String, List<Integer»> entryList = new ArrayList<Map.Entry<String, List<Integer»>(ranking.entrySet());
                Entry<String, List<Integer» lastEntry = entryList.get(entryList.size() - 1);
                List<Integer> subList = lastEntry.getValue();
                subList.set(1, 0);
                ranking.put(lastEntry.getKey(), subList);
                ranking.remove(lastEntry.getKey());

                // Redistribute votes
                List<List<Integer» checkBallot = new ArrayList<>();
                checkBallot = fileSystem.ballot;
                int check = fileSystem.candidates.indexOf(lastEntry.getKey());
                for(int i = 0; i < fileSystem.numOfVotes; i++) {
                    if(fileSystem.ballot.get(i).get(check) == initialRank) {
                        for(int j = 0; j < fileSystem.numOfCandidates; j++) {
                            if(fileSystem.ballot.get(i).get(j) == initialRank+1) {
                                List<Integer> subBallotList = ranking.get(fileSystem.candidates.get(j));
                                int oldValue = subBallotList.get(1);
                                oldValue++;
                                subBallotList.set(1, oldValue);
                                ranking.put(fileSystem.candidates.get(j), subBallotList);
                            }
                        }
                    }
                }
                ranking = checkRanking(ranking);
                entry = ranking.entrySet().iterator().next();
            }
            displayList.add(new LinkedHashMap<>(ranking));

        }

        // Displays error message if an error is detected
        else {
            JOptionPane.showMessageDialog(null,"The election type is not recognized.","ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

4.10 rankingsTest.java

```
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.*;

public class rankingsTest {


    @Test
    // This test case checks if checkRanking() sorts each candidate based on votes for CPL election
    public void test1_checkRankingsTest(){

        fileSystem.electionType = "CPL";
// set election type to IR
        fileSystem.numOfCandidates = 4;
// set number of parties to 4
        fileSystem.numOfVotes = 7;
// set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));    //
add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                              //
add 7 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));

        // expectedRanking is created and used to compare results with actualRanking to determine if
ranking is correct
        Map<String, List<Integer>> expectedRanking = new LinkedHashMap<>();
        expectedRanking.put("Republican", Arrays.asList(3,4));
        expectedRanking.put("Democratic", Arrays.asList(5,2));
        expectedRanking.put("Reform", Arrays.asList(6,1));
        expectedRanking.put("Green", Arrays.asList(7,0));

        // Create an object of rankings class and check ranking for all candidates
        rankings rank = new rankings();
        Map<String, List<Integer>> actualRanking = rank.checkRanking(countBallot.ballotWithName);

        // Test that the ranking for each candidate has been determined correctly
        assertEquals(expectedRanking, actualRanking);
    }


    @Test
    // This test case checks if checkRanking() sorts each candidate based on votes for IR election
    public void test2_checkRankingsTest(){

        fileSystem.electionType = "IR";                                                   // set
election type to IR
        fileSystem.numOfCandidates = 3;                                                   // set number
of candidates to 3
        fileSystem.numOfVotes = 4;                                                        // set number
of votes to 4
        fileSystem.candidates.add("Cedric Tan");                                        // add first
candidate
        fileSystem.candidates.add("Bryan Lee");                                         // add second
candidate
        fileSystem.candidates.add("Sherryl Ooi");                                       // add third
candidate
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));                               // add 4 arbitrary
ballots with rankings
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));

        // expectedRanking is created and used to compare results with actualRanking to determine if
ranking is correct
        Map<String, List<Integer>> expectedRanking = new LinkedHashMap<>();
        expectedRanking.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
        expectedRanking.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
        expectedRanking.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));

        // Create an object of rankings class and check ranking for all candidates
        rankings rank = new rankings();
        Map<String, List<Integer» actualRanking = rank.checkRanking(countBallot.ballotWithName);

        // Test that the ranking for each candidate has been determined correctly
        assertEquals(expectedRanking, actualRanking);
    }


    @Test
    // This test case checks if checkRanking() sorts each candidate based on votes for CPL election
    public void test3_checkRanking(){

        fileSystem.electionType = "CPL";                                                 // set election type to IR
        fileSystem.numOfCandidates = 4;                                                  // set number of parties to 4
        fileSystem.numOfVotes = 7;                                                       // set number of votes to 4
        fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green"));   // add parties
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));                                   // add 7 arbitrary ballots
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,0,1,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));
        fileSystem.ballot.add(Arrays.asList(1,0,0,0));
        fileSystem.ballot.add(Arrays.asList(0,1,0,0));

        // expectedResultsAfterCheckingMajority is created and used to compare with actual results after checking for Majority
        Map<String, List<Integer» expectedRanking = new LinkedHashMap<>();
        expectedRanking.put("Republican", Arrays.asList(3,4));
        expectedRanking.put("Democratic", Arrays.asList(5,2));
        expectedRanking.put("Reform", Arrays.asList(6,1));
        expectedRanking.put("Green", Arrays.asList(7,0));

        // Create an object of rankings class and check ranking for all candidates
        rankings rank = new rankings();
        Map<String, List<Integer» actualRanking = rank.checkRanking(countBallot.ballotWithName);

        // Test that the ranking for each candidate has been determined correctly after checking for Majority
        assertEquals(expectedRanking, actualRanking);
    }

    @Test
    // This test case checks if checkMajority() finds a majority among the candidates for IR election
    public void test4_checkMajority(){

        fileSystem.electionType = "IR";                                                  // set election type to IR
        fileSystem.numOfCandidates = 3;                                                  // set number of candidates to 3
        fileSystem.numOfVotes = 9;                                                       // set number of votes to 9
        fileSystem.candidates.add("Cedric Tan");                                         // add first candidate
        fileSystem.candidates.add("Bryan Lee");                                          // add second candidate
        fileSystem.candidates.add("Sherryl Ooi");                                        // add third candidate
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));                                   // add 9 arbitrary ballots with rankings
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));
        fileSystem.ballot.add(Arrays.asList(2, 1, 3));
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));

        // expectedResultsAfterCheckingMajority is created and used to compare with actual results after checking for Majority
        Map<String, List<Integer» expectedResultsAfterCheckingMajority = new LinkedHashMap<>();
        expectedResultsAfterCheckingMajority.put("Sherryl Ooi", Arrays.asList(0, 0, 3, 3));
        expectedResultsAfterCheckingMajority.put("Bryan Lee", Arrays.asList(0, 6, 3, 3));
        expectedResultsAfterCheckingMajority.put("Cedric Tan", Arrays.asList(0, 0, 3, 3));

        // Create an object of rankings class and check for majority
        rankings rank = new rankings();
        rank.checkMajority(countBallot.ballotWithName);

        // Test that the ranking for each candidate has been determined correctly after checking for
```

```
Majority
        assertEquals(expectedResultsAfterCheckingMajority, countBallot.ballotWithName);
    }

    @Test
    // This test case checks if checkMajority() finds a majority among the candidates for IR election
    public void test5_checkMajority(){

        fileSystem.electionType = "IR";                                                // set election type to IR
        fileSystem.numOfCandidates = 3;                                                // set number of candidates to 3
        fileSystem.numOfVotes = 4;                                                     // set number of votes to 4
        fileSystem.candidates.add("Cedric Tan");                                      // add first candidate
        fileSystem.candidates.add("Bryan Lee");                                       // add second candidate
        fileSystem.candidates.add("Sherryl Ooi");                                     // add third candidate
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));                               // add 4 arbitrary ballots with rankings
        fileSystem.ballot.add(Arrays.asList(2, 3, 1));
        fileSystem.ballot.add(Arrays.asList(3, 2, 1));
        fileSystem.ballot.add(Arrays.asList(1, 3, 2));

        // expectedResultsAfterCheckingMajority is created and used to compare with actual results after checking for Majority
        Map<String, List<Integer» expectedResultsAfterCheckingMajority = new LinkedHashMap<>();
        expectedResultsAfterCheckingMajority.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
        expectedResultsAfterCheckingMajority.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
        expectedResultsAfterCheckingMajority.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));

        // Create an object of rankings class and check for majority
        rankings rank = new rankings();
        rank.checkMajority(countBallot.ballotWithName);

        // Test that the ranking for each candidate has been determined correctly after checking for Majority
        assertEquals(expectedResultsAfterCheckingMajority, countBallot.ballotWithName);
    }
}
```

4.11 votingSystem.java

```
public class votingSystem {
    public static void main (String args[]) {
        displayResults results = new displayResults();
    }
}
```

Index