

Voting System Software

Generated by Doxygen 1.9.6

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 countBallot Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 countBallot()	6
3.2 countBallotTest Class Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Function Documentation	6
3.2.2.1 test1_countBallot()	7
3.2.2.2 test2_countBallot()	7
3.2.2.3 test3_countBallot()	7
3.3 displayResults Class Reference	7
3.3.1 Detailed Description	7
3.3.2 Constructor & Destructor Documentation	8
3.3.2.1 displayResults()	8
3.3.3 Member Function Documentation	8
3.3.3.1 generateAuditFile()	8
3.3.3.2 showResults()	8
3.4 displayResultsTest Class Reference	8
3.4.1 Detailed Description	9
3.4.2 Member Function Documentation	9
3.4.2.1 test1_generateAuditFile()	9
3.4.2.2 test2_generateAuditFile()	9
3.4.2.3 test3_showResults()	9
3.4.2.4 test4_showResults()	10
3.5 fileSystem Class Reference	10
3.5.1 Detailed Description	10
3.5.2 Constructor & Destructor Documentation	10
3.5.2.1 fileSystem()	11
3.5.3 Member Function Documentation	11
3.5.3.1 checkFileFormat()	11
3.5.3.2 fileSystemRead()	11
3.5.3.3 getFileExtension()	11
3.5.3.4 openFile()	12
3.5.3.5 readFile()	12
3.6 fileSystemTest Class Reference	13
3.6.1 Detailed Description	13

3.6.2 Member Function Documentation	13
3.6.2.1 test10_checkFileFormat()	13
3.6.2.2 test11_readFile()	14
3.6.2.3 test12_readFile()	14
3.6.2.4 test13_openFile()	14
3.6.2.5 test14_openFile()	14
3.6.2.6 test1_getFileExtension()	14
3.6.2.7 test2_getFileExtension()	14
3.6.2.8 test3_getFileExtension()	15
3.6.2.9 test4_getFileExtension()	15
3.6.2.10 test5_getFileExtension()	15
3.6.2.11 test6_getFileExtension()	15
3.6.2.12 test7_checkFileFormat()	15
3.6.2.13 test8_checkFileFormat()	15
3.6.2.14 test9_checkFileFormat()	16
3.7 finalRanking Class Reference	16
3.7.1 Detailed Description	16
3.7.2 Constructor & Destructor Documentation	16
3.7.2.1 finalRanking()	16
3.7.3 Member Function Documentation	17
3.7.3.1 checkForTie()	17
3.7.3.2 fairCoinToss()	17
3.7.3.3 poolCoinToss()	17
3.8 finalRankingTest Class Reference	18
3.8.1 Detailed Description	18
3.8.2 Member Function Documentation	18
3.8.2.1 test1_checkForTie()	18
3.8.2.2 test2_checkForTie()	19
3.8.2.3 test3_checkForTie()	19
3.8.2.4 test4_fairCoinToss()	19
3.8.2.5 test5_poolCoinToss()	19
3.9 rankings Class Reference	19
3.9.1 Detailed Description	20
3.9.2 Constructor & Destructor Documentation	20
3.9.2.1 rankings()	20
3.9.3 Member Function Documentation	20
3.9.3.1 checkMajority()	20
3.9.3.2 checkRanking()	21
3.10 rankingsTest Class Reference	21
3.10.1 Detailed Description	21
3.10.2 Member Function Documentation	22
3.10.2.1 test1_checkRankingsTest()	22

3.10.2.2 test2_checkRankingsTest()	22
3.10.2.3 test3_checkRanking()	22
3.10.2.4 test4_checkMajority()	22
3.10.2.5 test5_checkMajority()	22
3.11 votingSystem Class Reference	23
3.11.1 Detailed Description	23
3.11.2 Member Function Documentation	23
3.11.2.1 main()	23
4 File Documentation	25
4.1 countBallot.java	25
4.2 countBallotTest.java	26
4.3 displayResults.java	27
4.4 displayResultsTest.java	29
4.5 fileSystem.java	32
4.6 fileSystemTest.java	34
4.7 finalRanking.java	37
4.8 finalRankingTest.java	39
4.9 rankings.java	42
4.10 rankingsTest.java	44
4.11 votingSystem.java	46
Index	47

Chapter 1

Class Index

1.1 Class List

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

countBallot	5
countBallotTest	6
displayResults	7
displayResultsTest	8
fileSystem	10
fileSystemTest	13
finalRanking	16
finalRankingTest	18
rankings	19
rankingsTest	21
votingSystem	23

Chapter 2

File Index

2.1 File List

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

src/ countBallot.java	25
src/ countBallotTest.java	26
src/ displayResults.java	27
src/ displayResultsTest.java	29
src/ fileSystem.java	32
src/ fileSystemTest.java	34
src/ finalRanking.java	37
src/ finalRankingTest.java	39
src/ rankings.java	42
src/ rankingsTest.java	44
src/ votingSystem.java	46

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
SherryI 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
SherryI 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

<i>fileName</i>	- 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

<i>fullName</i>	- 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

<i>fileName</i>	- 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

<i>range</i>	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
SherryI 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

<i>totalBallots</i>	— 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

<i>newBallots</i>	— a 2D array indicating the number of votes for each candidate
<i>initialRank</i>	- 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
SherryI 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

```
00001 import javax.swing.*;
00002 import java.util.*;
00003
00016 public class countBallot {
00017
00018     // An array list that stores the count of total ballots for each candidate
00019     static List<List<Integer>> totalBallots = new ArrayList<>();
00020
00021     // A HashMap that stores the count of total ballots for each candidate along with the candidate
00022     name static Map <String, List<Integer>> ballotWithName = new HashMap<>();
00023
00028     public countBallot () {
00029
00030         // Create fileSystem object
00031         fileSystem files = new fileSystem();
00032
00033         // If the election type is Closed Party Listing (CPL)
00034         if(files.electionType.equals("CPL")) {
00035
00036             for(int i = 0; i < files.numOfCandidates; i++) {
00037                 List<Integer> subBallotCount = new ArrayList<>();
00038                 subBallotCount.add(0);
00039                 subBallotCount.add(0);
00040                 totalBallots.add(subBallotCount);
00041             }
00042
00043             // Loops through number of votes and candidates to allocate ballots based on parties
00044             for (int i = 0; i < files.numOfCandidates; i++) {
00045                 for (int j = 0; j < files.numOfVotes; j++) {
00046
00047                     // if the ballot received by the party is 1
00048                     if(files.ballot.get(j).get(i) == 1) {
00049                         int oldValue = totalBallots.get(i).get(1);
00050                         oldValue++; // increment the count by 1
00051                         totalBallots.get(i).set(1, oldValue); // update the number of 1s the party
00052                             received
00053                             }
00054
00055                             // if the ballot received by the party is 0
00056                             else {
00057                                 int oldValue = totalBallots.get(i).get(0);
00058                                 oldValue++; // increment the count by 1
00059                                 totalBallots.get(i).set(0, oldValue); // update the total number of 0s the
00060                                     party received
00061                                     }
00062                                     }
00063                                     }
00064
00065                             // If the election type is Instant Runoff Voting (IR)
00066                             else if(files.electionType.equals("IR")){
00067
00068                                 for(int i = 0; i < files.numOfCandidates; i++) {
00069                                     List<Integer> subBallotCount = new ArrayList<>();
00070                                     for(int j = 0; j <= files.numOfCandidates; j++) {
00071                                         subBallotCount.add(0);
00072                                     }
00073                                 }
00074                             }
00075                         }
```

```

00072         totalBallots.add(subBallotCount);
00073     }
00074
00075     // Loops through number of votes and candidates to allocate the ballots based on the
    ranking of each candidate
00076     for (int i = 0; i < fileSystem.numOfCandidates; i++) {
00077         for (int j = 0; j < fileSystem.numOfVotes; j++) {
00078             for (int k = 0; k <= fileSystem.numOfCandidates; k++) {
00079                 if(fileSystem.ballot.get(j).get(i) == k) {
00080                     int oldValue = totalBallots.get(i).get(k);
00081                     oldValue++; // increment the count by 1
00082                     totalBallots.get(i).set(k, oldValue); // update the total number of
    ballots the candidate received
00083                 }
00084             }
00085         }
00086     }
00087 }
00088
00089 // If the election type is neither IR nor CPL
00090 else {
00091     JOptionPane.showMessageDialog(null,"The election type is not recognized.,"ERROR
    MESSAGE",JOptionPane.ERROR_MESSAGE);
00092 }
00093
00094 for(int i = 0; i < fileSystem.numOfCandidates; i++) {
00095     ballotWithName.put(fileSystem.candidates.get(i), totalBallots.get(i));
00096 }
00097 }
00098 }
00099 }

```

4.2 countBallotTest.java

```

00001 import static org.junit.Assert.*;
00002 import org.junit.Test;
00003 import java.util.*;
00004
00018 public class countBallotTest {
00019
00020     @Test
00021     // This test checks that countBallot() returns the correct TotalBallots list and ballotWithName
    map for CPL
00022     public void test1_countBallot(){
00023
00024         fileSystem.electionType = "CPL";
00025         // set election type to IR
00026         fileSystem.numOfCandidates = 4;
00027         // set number of parties to 4
00028         fileSystem.numOfVotes = 7;
00029         // set number of votes to 4
00030         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
    add parties
00031         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
    add 7 arbitrary ballots
00032         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00033         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00034         fileSystem.ballot.add(Arrays.asList(0,0,0,1));
00035         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00036         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00037         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00038
00039         // Create an object of the countBallot class to count totalBallots and ballotsWithName
00040         countBallot ballotCount = new countBallot();
00041
00042         // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
00043         List<List<Integer>> expectedTotalBallots = Arrays.asList(
00044             // (index 0: number of zeros in all ballots, index 1: number of ones in all ballots)
00045             Arrays.asList(5,2),
00046             Arrays.asList(4,3),
00047             Arrays.asList(6,1),
00048             Arrays.asList(6,1)
00049         );
00050         // Test that the totalBallots list has been calculated correctly using assertEquals
00051         assertEquals(expectedTotalBallots, ballotCount.totalBallots);
00052
00053         // expectedBallotWithName is created and used to compare results with
    ballotCount.ballotWithName
00054         Map<String, List<Integer>> expectedBallotWithName = new HashMap<>();
00055         expectedBallotWithName.put("Democratic", Arrays.asList(5,2));
00056         expectedBallotWithName.put("Republican", Arrays.asList(4,3));
00057         expectedBallotWithName.put("Reform", Arrays.asList(6,1));
00058         expectedBallotWithName.put("Green", Arrays.asList(6,1));

```



```

00056         // Test that the ballotWithName list has been calculated correctly using assertEquals
00057         assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
00058     }
00059
00060
00061     @Test
00062     // The second test checks that countBallot() returns the correct TotalBallots list and
    ballotWithName map for IR
00063     public void test2_countBallot(){
00064
00065         fileSystem.electionType = "IR"; // set
00066         election type to IR
00066         fileSystem.numOfCandidates = 3; // set number
00066         of candidates to 3
00067         fileSystem.numOfVotes = 4; // set number
00067         of votes to 4
00068         fileSystem.candidates.add("Sherryl Ooi"); // add first
00068         candidate
00069         fileSystem.candidates.add("Bryan Lee"); // add second
00069         candidate
00070         fileSystem.candidates.add("Cedric Tan"); // add third
00070         candidate
00071         fileSystem.ballot.add(Arrays.asList(1, 2, 3)); // add 4 arbitrary
00071         ballots with rankings
00072         fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00073         fileSystem.ballot.add(Arrays.asList(1, 2, 3));
00074         fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00075
00076         // Create an object of the countBallot class to count totalBallots and ballotsWithName
00077         countBallot ballotCount = new countBallot();
00078
00079         // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
00080         List<List<Integer>> expectedTotalBallots = Arrays.asList(
00081             // (index 0: number of zeros in all ballots, index 1: number of ones in all ballots)
00082             Arrays.asList(0, 3, 1, 0),
00083             Arrays.asList(0, 0, 2, 2),
00084             Arrays.asList(0, 1, 1, 2)
00085         );
00086         // Test that the totalBallots list has been calculated correctly using assertEquals
00087         assertEquals(expectedTotalBallots, ballotCount.totalBallots);
00088
00089         // expectedBallotWithName is created and used to compare results with
    ballotCount.ballotWithName
00090         Map<String, List<Integer>> expectedBallotWithName = new HashMap<>();
00091         expectedBallotWithName.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
00092         expectedBallotWithName.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
00093         expectedBallotWithName.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));
00094         // Test that the ballotWithName list has been calculated correctly using assertEquals
00095         assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
00096     }
00097
00098
00099     @Test
00100     // This test checks that countBallot() returns an empty TotalBallots list and ballotWithName map
    when election type is neither CPL or IR
00101     public void test3_countBallot(){
00102
00103         fileSystem.electionType = "OPL"; // set election type to Open Party List
00104
00105         // expectedTotalBallots is created and used to compare results with ballotCount.totalBallots
00106         List<List<Integer>> expectedTotalBallots = Arrays.asList();
00107         countBallot ballotCount = new countBallot();
00108         // Invalid election type means that ballots will not be counted therefore the array list is
    empty
00109         assertEquals(expectedTotalBallots, ballotCount.totalBallots);
00110
00111         // expectedBallotWithName is created and used to compare results with
    ballotCount.ballotWithName
00112         Map<String, List<Integer>> expectedBallotWithName = new HashMap<>();
00113         // Invalid election type means that ballots with name will not be counted therefore the map is
    empty
00114         assertEquals(expectedBallotWithName, ballotCount.ballotWithName);
00115     }
00116
00117 }
00118
00119

```

4.3 displayResults.java

```

00001 import java.io.*;
00002 import javax.swing.*;
00003 import java.util.*;

```

```

00004
00017 public class displayResults {
00018
00023     public displayResults () {
00024         finalRanking rank = new finalRanking();
00025         // showResults();
00026     }
00027
00032     public void generateAuditFile () {
00033         try {
00034
00035             // Create JFrame to display information
00036             JFrame parentFrame = new JFrame();
00037             JFileChooser fileChooser = new JFileChooser();
00038             fileChooser.setDialogTitle("Save audit file");
00039
00040             int userSelection = fileChooser.showSaveDialog(parentFrame);
00041
00042             // Checks whether approve (yes, ok) is chosen
00043             if (userSelection == JFileChooser.APPROVE_OPTION) {
00044                 File outputFile = fileChooser.getSelectedFile();
00045                 System.out.println("Save as file: " + outputFile.getAbsolutePath());
00046
00047                 // Generate output file
00048                 outputFile.createNewFile();
00049                 PrintWriter output = new PrintWriter(outputFile);
00050                 StringBuffer csvData = new StringBuffer("");
00051
00052                 csvData.append("Election type: " + fileSystem.electionType + "\n");
00053
00054                 // If the election type is Closed Party Listing (CPL)
00055                 if(fileSystem.electionType.equals("CPL")) {
00056                     // add election information into the audit file
00057                     csvData.append("Number of Parties: " + fileSystem.numOfCandidates + "\n");
00058                     csvData.append("Parties joined election: " + fileSystem.candidates + "\n");
00059                     for(int i = 0; i < fileSystem.numOfCandidates; i++) {
00060                         csvData.append("Candidates of party " + fileSystem.candidates.get(i) + ": " +
00061                             fileSystem.candidatesList.get(i) + "\n");
00062                     }
00063                     csvData.append("Total seats elected: " + fileSystem.numOfSeats + "\n");
00064                     csvData.append("Total number of voters: " + fileSystem.numOfVotes + "\n");
00065
00066                     // add ranking for each candidate into GUI
00067                     List<Map.Entry<String, List<Integer>> entries = new
00068                     ArrayList<>(rankings.ranking.entrySet());
00069                     List<String> keys = new ArrayList<>(rankings.ranking.keySet());
00070                     for(int i = 0; i < fileSystem.numOfCandidates; i++) {
00071                         csvData.append(entries.get(i).getKey() + " is rank " +
00072                             (keys.indexOf(entries.get(i).getKey()) + 1) + ".\n");
00073                     }
00074                     csvData.append("The final winners of the election are parties: \n");
00075
00076                     // add final winners' ranking into the GUI
00077                     List<Map.Entry<String, List<Integer>> entries2 = new
00078                     ArrayList<>(finalRanking.finalRanking.entrySet());
00079                     List<String> keys2 = new ArrayList<>(finalRanking.finalRanking.keySet());
00080                     for(int i = 0; i < fileSystem.numOfSeats; i++) {
00081                         csvData.append(entries2.get(i).getKey() + " is rank " +
00082                             (keys2.indexOf(entries2.get(i).getKey()) + 1) + ".\n");
00083                     }
00084
00085                     // If the election type is Instant Runoff Voting (IR)
00086                     else if(fileSystem.electionType.equals("IR")){
00087                         // add election information into the audit file
00088                         csvData.append("Number of Candidates: " + fileSystem.numOfCandidates + "\n");
00089                         csvData.append("Candidates joined election: " + fileSystem.candidates + "\n");
00090                         csvData.append("Total number of voters: " + fileSystem.numOfVotes + "\n");
00091
00092                         // add the final winner and displays the results below the final winner
00093                         List<Map.Entry<String, List<Integer>> entries = new
00094                         ArrayList<>(rankings.ranking.entrySet());
00095                         csvData.append("The final winner of the election is candidate " +
00096                             entries.get(0).getKey() + "\n");
00097                         csvData.append("\nThe results are as below: \n");
00098                         for(int i = 0; i < rankings.displayList.size(); i++) {
00099                             csvData.append(rankings.displayList.get(i) + "\n");
00100                         }
00101                     }
00102
00103                     output.write(csvData.toString());
00104                     output.close();
00105                 }
00106             }
00107         } catch (Exception e) {
00108             e.printStackTrace();
00109         }
00110     }
00111 }

```

```

00104     }
00105 }
00106
00111 public void showResults () {
00112     // GUI interface to prompt user to choose whether to generate an audit file
00113     int yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to generate Audit
File?", "Generate Audit File",JOptionPane.YES_NO_OPTION);
00114
00115     // If the user chose YES to generate an audit file
00116     if(yesOrNo == JOptionPane.YES_OPTION) {
00117         generateAuditFile();
00118     }
00119
00120     // Create JFrame to display information
00121     JFrame frame = new JFrame ("Final Results");
00122     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
00123     frame.setSize(800, 800);
00124     JTextArea textArea = new JTextArea();
00125
00126     // Adds the election type onto the first line of the JFrame
00127     textArea.append("Election type: " + fileSystem.electionType + "\n");
00128
00129     // If the election type is Closed Party Listing (CPL)
00130     if(fileSystem.electionType.equals("CPL")) {
00131         // add election information into the GUI
00132         textArea.append("Number of Parties: " + fileSystem.numOfCandidates + "\n");
00133         textArea.append("Parties joined election: " + fileSystem.candidates + "\n");
00134         for(int i = 0; i < fileSystem.numOfCandidates; i++) {
00135             textArea.append("Candidates of party " + fileSystem.candidates.get(i) + ": " +
fileSystem.candidatesList.get(i) + "\n");
00136         }
00137         textArea.append("Total seats elected: " + fileSystem.numOfSeats + "\n");
00138         textArea.append("Total number of voters: " + fileSystem.numOfVotes + "\n");
00139
00140         // add final winners into the GUI
00141         List<Map.Entry<String, List<Integer>>> entries = new
ArrayList<>(rankings.ranking.entrySet());
00142         List<String> keys = new ArrayList<>(rankings.ranking.keySet());
00143         for(int i = 0; i < fileSystem.numOfCandidates; i++) {
00144             textArea.append(entries.get(i).getKey() + " is rank " +
(keys.indexOf(entries.get(i).getKey()) + 1) + ".\n");
00145         }
00146         textArea.append("The final winners of the election are parties: \n");
00147
00148         // add final winners' ranking into the GUI
00149         List<Map.Entry<String, List<Integer>>> entries2 = new
ArrayList<>(finalRanking.finalRanking.entrySet());
00150         List<String> keys2 = new ArrayList<>(finalRanking.finalRanking.keySet());
00151         for(int i = 0; i < fileSystem.numOfSeats; i++) {
00152             textArea.append(entries2.get(i).getKey() + " is rank " +
(keys2.indexOf(entries2.get(i).getKey()) + 1) + ".\n");
00153         }
00154     }
00155
00156     // If the election type is Instant Runoff Voting (IR)
00157     else if(fileSystem.electionType.equals("IR")){
00158         // add election information into the GUI
00159         textArea.append("Number of Candidates: " + fileSystem.numOfCandidates + "\n");
00160         textArea.append("Candidates joined election: " + fileSystem.candidates + "\n");
00161         textArea.append("Total number of voters: " + fileSystem.numOfVotes + "\n");
00162
00163         // add the final winner and displays the results below the final winner
00164         List<Map.Entry<String, List<Integer>>> entries = new
ArrayList<>(rankings.ranking.entrySet());
00165         textArea.append("The final winner of the election is candidate " +
entries.get(0).getKey() + "\n");
00166         textArea.append("\nThe results are as below: \n");
00167         for(int i = 0; i < rankings.displayList.size(); i++) {
00168             textArea.append(rankings.displayList.get(i) + "\n");
00169         }
00170     }
00171
00172     // modifications to JFrame
00173     frame.add(textArea);
00174     textArea.setEditable(false);
00175     frame.setLocationRelativeTo(null);
00176     frame.setVisible(true);
00177 }
00178 }

```

4.4 displayResultsTest.java

```
00001 import org.junit.Test;
```

```

00002 import static org.junit.Assert.*;
00003 import java.util.*;
00004
00019 public class displayResultsTest {
00020
00021     @Test
00022     // This test case checks if generateAuditFile() is called to export an audit file when the
    election type is CPL
00023     public void test1_generateAuditFile(){
00024
00025         fileSystem.electionType = "CPL";
00026         // set election type to IR
00027         fileSystem.numOfCandidates = 4;
00028         // set number of parties to 4
00029         fileSystem.numOfSeats = 5;
00030         // set number of seats to 5
00031         fileSystem.numOfVotes = 5;
00032         // set number of votes to 5
00033         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
    add parties
00034         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
    add 5 arbitrary ballots
00035         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00036         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00037         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00038         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00039
00040         // Creates a new object of the displayResults class
00041         displayResults dR = new displayResults();
00042
00043         // +-----+
00044         // | Results: |
00045         // +-----+
00046         // | "Democratic", (3,2) |
00047         // | "Republican", (2,3) |
00048         // | "Reform", (5,0) |
00049         // | "Green", (5,0) |
00050         // +-----+
00051
00052         // Initialize boolean
00053         boolean CheckIfAuditFileIsGenerated = true;
00054         // Checks if the generateAuditFile() method is called using try and catch
00055         try {
00056             dR.generateAuditFile();
00057             dR.showResults();
00058         }
00059         catch (Exception e) { CheckIfAuditFileIsGenerated = false; }
00060         assertFalse(CheckIfAuditFileIsGenerated);
00061
00062     @Test
00063     // This test case checks if generateAuditFile() is called to export an audit file when the
    election type is IR
00064     public void test2_generateAuditFile(){
00065
00066         fileSystem.electionType = "IR"; // set
    election type to IR
00067         fileSystem.numOfCandidates = 3; // set number
    of candidates to 3
00068         fileSystem.numOfVotes = 5; // set number
    of votes to 5
00069         fileSystem.candidates.add("Cedric Tan"); // add first
    candidate
00070         fileSystem.candidates.add("Bryan Lee"); // add second
    candidate
00071         fileSystem.candidates.add("Sherryl Ooi"); // add third
    candidate
00072         fileSystem.ballot.add(Arrays.asList(2, 1, 3)); // add 5 arbitrary
    ballots with rankings
00073         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00074         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00075         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00076         fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00077
00078         // Creates a new object of the displayResults class
00079         displayResults dR = new displayResults();
00080
00081         // +-----+
00082         // | Results: |
00083         // +-----+
00084         // | "Cedric", (0,0,2,3) |
00085         // | "Bryan", (0,1,3,1) |
00086         // | "Sherryl", (0,4,0,1) |
00087         // +-----+
00088
00089         // Initialize boolean

```

```

00088         boolean CheckIfAuditFileIsGenerated = true;
00089         // Checks if the generateAuditFile() method is called using try and catch
00090         try {
00091             dR.generateAuditFile();
00092             dR.showResults();
00093         }
00094         catch (Exception e) { CheckIfAuditFileIsGenerated = false; }
00095         assertTrue(CheckIfAuditFileIsGenerated);
00096     }
00097
00098
00099     @Test
00100     // This test case checks if showResults() is called to display the election results when the
    election type is IR
00101     public void test3_showResults(){
00102
00103         fileSystem.electionType = "CPL";
00104         // set election type to IR
00105         fileSystem.numOfCandidates = 4;
00106         // set number of parties to 4
00107         fileSystem.numOfSeats = 6;
00108         // set number of seats to 4
00109         fileSystem.numOfVotes = 5;
00110         // set number of votes to 5
00111         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
    add parties
00112         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
    add 5 arbitrary ballots
00113         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00114         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00115         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00116         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00117
00118         // Creates a new object of the displayResults class
00119         displayResults dR = new displayResults();
00120
00121         // +-----+
00122         // | Results: |
00123         // +-----+
00124         // | "Democratic", (3,2) |
00125         // | "Republican", (2,3) |
00126         // | "Reform", (5,0) |
00127         // | "Green", (5,0) |
00128         // +-----+
00129
00130         // Initialize boolean
00131         boolean CheckIfResultsAreDisplayed = true;
00132         // Checks if the showResults() method is called using try and catch
00133         try { dR.showResults();
00134             dR.generateAuditFile(); } catch (Exception e) { CheckIfResultsAreDisplayed = false; }
00135         assertFalse(CheckIfResultsAreDisplayed);
00136     }
00137
00138
00139     @Test
00140     // This test case checks if showResults() is called to display the election results when the
    election type is IR
00141     public void test4_showResults(){
00142
00143         fileSystem.electionType = "IR"; // set
    election type to IR
00144         fileSystem.numOfCandidates = 3; // set number
    of candidates to 3
00145         fileSystem.numOfVotes = 5; // set number
    of votes to 5
00146         fileSystem.candidates.add("Cedric Tan"); // add first
    candidate
00147         fileSystem.candidates.add("Bryan Lee"); // add second
    candidate
00148         fileSystem.candidates.add("Sherryl Ooi"); // add third
    candidate
00149         fileSystem.ballot.add(Arrays.asList(2, 1, 3)); // add 5 arbitrary
    ballots with rankings
00150         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00151         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00152         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00153         fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00154
00155         // Creates a new object of the displayResults class
00156         displayResults dR = new displayResults();
00157
00158         // +-----+
00159         // | Results: |
00160         // +-----+
00161         // | "Cedric", (0,0,2,3) |
00162         // | "Bryan", (0,1,3,1) |
00163         // | "Sherryl", (0,4,0,1) |

```

```

00160         // +-----+
00161
00162         // Initialize boolean
00163         boolean CheckIfResultsAreDisplayed= true;
00164         // Checks if the showResults() method is called using try and catch
00165         try { dR.showResults(); } catch (Exception e) { CheckIfResultsAreDisplayed = false; }
00166         assertTrue(CheckIfResultsAreDisplayed);
00167     }
00168
00169 }
00170

```

4.5 fileSystem.java

```

00001 import java.io.*;
00002 import java.util.*;
00003 import javax.swing.*;
00004 import javax.swing.JFileChooser;
00005
00016 public class fileSystem {
00017
00018     // Instance variables that are used in fileSystem class
00019     static String electionType;
00020     static int numofCandidates;
00021     static List<String> candidates = new ArrayList<>();
00022     static int numofSeats;
00023     static int numofVotes;
00024     static List<List<Integer>> ballot = new ArrayList<>();
00025     static List<List<String>> candidatesList = new ArrayList<>();
00026     static int fileCount = 0;
00027
00028
00034     public fileSystem() {
00035         fileSystemRead();
00036         int yesOrNo = JOptionPane.showConfirmDialog(null,"Do you have other ballot files","Multiple
file",JOptionPane.YES_NO_OPTION);
00037         while (yesOrNo == JOptionPane.YES_OPTION) {
00038             fileSystemRead();
00039             yesOrNo = JOptionPane.showConfirmDialog(null,"Do you have other ballot files","Multiple
file",JOptionPane.YES_NO_OPTION);
00040         }
00041         JOptionPane.showMessageDialog(null,"You have a total of " + fileCount + " files opened.,"INFO
MESSAGE",JOptionPane.INFORMATION_MESSAGE);
00042     }
00043
00044     public static void fileSystemRead() {
00045         JFileChooser fileChooser = new JFileChooser();
00046         int result = fileChooser.showOpenDialog(null);
00047
00048         // Checks selected file to see if approve (yes, ok) is chosen
00049         if (result == JFileChooser.APPROVE_OPTION) {
00050             File selectedFile = fileChooser.getSelectedFile();
00051
00052             // Opens selected file
00053             openFile(selectedFile);
00054         }
00055
00056         else {
00057             JOptionPane.showMessageDialog(null,"There is an error opening the file.,"ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
00058         }
00059     }
00060
00067     public static void openFile(File fileName) {
00068
00069         // checks if file format is correct
00070         if (checkFileFormat(fileName)) {
00071             JOptionPane.showMessageDialog(null,"The file is open.,"INFO
MESSAGE",JOptionPane.INFORMATION_MESSAGE);
00072             readFile(fileName);
00073         }
00074
00075         // Displays error message if an error is detected
00076         else {
00077             JOptionPane.showMessageDialog(null,"There is an error opening the file.,"ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
00078         }
00079     }
00080
00087     public static boolean checkFileFormat (File fileName) {
00088
00089         // checks if file type is a .csv file type
00090         if (getFileExtension(fileName).equals("csv")) {

```

```

00091         return true;
00092     }
00093
00094     if (fileName == null) {
00095         return false;
00096     }
00097
00098     return false;
00099 }
00100
00105 public static void readFile (File fileName) {
00106
00107     // Read lines inside CSV file
00108     try {
00109
00110         // Variables to store information that is read from CSV file
00111         if(fileCount == 0) {
00112             // They read the first file
00113             BufferedReader file = new BufferedReader(new FileReader(fileName));
00114             electionType = file.readLine();
00115             numOfCandidates = Integer.parseInt(file.readLine());
00116             // allCandidates = file.readLine();
00117             // String[] subCandidates = allCandidates.split(", ");
00118             candidates.addAll(Arrays.asList(file.readLine().split(", ")));
00119
00120             // Read in values based on election type
00121             if(electionType.equals("CPL")) {
00122                 for(int i = 0; i < numOfCandidates; i++) {
00123                     // List<String> subCandidatesList = new ArrayList<>(file.readLine());
00124                     candidatesList.add(i, Arrays.asList(file.readLine()));
00125                 }
00126                 numOfSeats = Integer.parseInt(file.readLine());
00127                 numOfVotes = Integer.parseInt(file.readLine());
00128             }
00129
00130             else if (electionType.equals("IR")) {
00131                 numOfVotes = Integer.parseInt(file.readLine());
00132             }
00133
00134             // Invalid election type
00135             else {
00136                 JOptionPane.showMessageDialog(null,"The election type is not recognized.", "ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
00137             }
00138
00139             String CurrentLine;
00140
00141             // While loop to read in number of candidates from CSV file
00142             while ((CurrentLine = file.readLine()) != null)
00143             {
00144                 // Splits the line into array of strings based on commas
00145                 String [] fileLine = CurrentLine.split(",", -1);
00146                 List<Integer> subBallot = new ArrayList<>();
00147                 // Adds the candidates' ranking into ballot array
00148                 for(int i = 0; i < numOfCandidates; i++) {
00149                     if(fileLine[i] != "") {
00150                         subBallot.add(Integer.parseInt(fileLine[i]));
00151                     }
00152                     else {
00153                         subBallot.add(0);
00154                     }
00155                 }
00156                 ballot.add(subBallot);
00157             }
00158
00159             // Close file
00160             file.close();
00161
00162             // Update the count variable for the number of files opened
00163             fileCount ++;
00164         }
00165         else if(fileCount > 0) {
00166             // Read other files
00167             BufferedReader file = new BufferedReader(new FileReader(fileName));
00168
00169             // readLine of new file to check if the header is identical
00170             String electionTypeNew = file.readLine();
00171             int numOfCandidatesNew = Integer.parseInt(file.readLine());
00172             List<String> candidatesNew = new ArrayList<>();
00173             candidatesNew.addAll(Arrays.asList(file.readLine().split(", ")));
00174
00175             if(electionTypeNew.equals(electionType) && numOfCandidates == numOfCandidatesNew &&
candidatesNew.equals(candidates))
00176             {
00177                 // Read in values based on election type
00178                 // If the election type is Closed Party Listing (CPL)
00179                 if(electionType.equals("CPL")) {

```

```

00180         List<List<String>> candidatesListNew = new ArrayList<>();
00181         for(int i = 0; i < numOfCandidates; i++) {
00182             candidatesListNew.add(i, Arrays.asList(file.readLine()));
00183         }
00184         int numOfSeatsNew = Integer.parseInt(file.readLine());
00185         int numOfVotesNew = Integer.parseInt(file.readLine());
00186     }
00187
00188     // If the election type is Instant Runoff Voting (IR)
00189     else if (electionType.equals("IR")) {
00190         int numOfVotesNew = Integer.parseInt(file.readLine());
00191     }
00192
00193     // If the election type is invalid
00194     else {
00195         JOptionPane.showMessageDialog(null, "The election type is not
recognized.", "ERROR MESSAGE", JOptionPane.ERROR_MESSAGE);
00196     }
00197
00198     String CurrentLine;
00199
00200     // While loop to read in number of candidates from CSV file
00201     while ((CurrentLine = file.readLine()) != null)
00202     {
00203         // Splits the line into array of strings based on commas
00204         String [] fileLine = CurrentLine.split(", ", -1);
00205         List<Integer> subBallot = new ArrayList<>();
00206         // Adds the candidates' ranking into ballot array
00207         for(int i = 0; i < numOfCandidates; i++) {
00208             if(fileLine[i] != "") {
00209                 subBallot.add(Integer.parseInt(fileLine[i]));
00210             }
00211             else {
00212                 subBallot.add(0);
00213             }
00214         }
00215         ballot.add(subBallot);
00216     }
00217     // Close file
00218     file.close();
00219
00220     // Update the count variable for the number of files opened
00221     fileCount++;
00222 }
00223 else {
00224     JOptionPane.showMessageDialog(null, "The file does not have the same
header.", "ERROR MESSAGE", JOptionPane.ERROR_MESSAGE);
00225 }
00226 }
00227 }
00228
00229 // Displays error message if an error is detected
00230 catch (Exception e) {
00231     JOptionPane.showMessageDialog(null, "There is an error opening the file.", "ERROR
MESSAGE", JOptionPane.ERROR_MESSAGE);
00232     e.printStackTrace();
00233 }
00234 }
00235 }
00236
00242 public static String getFileExtension(File fullName) {
00243
00244     String fileName = fullName.getName();
00245     int dotIndex = fileName.lastIndexOf('.');
00246
00247     // Returns string indicating file type after "." in file name
00248     return (dotIndex == -1) ? "" : fileName.substring(dotIndex + 1);
00249 }
00250 }

```

4.6 fileSystemTest.java

```

00001 import static org.junit.Assert.*;
00002 import org.junit.Test;
00003 import java.io.*;
00004 import java.util.*;
00005
00018 public class fileSystemTest {
00019
00020     @Test
00021     // This test checks that getFileExtension() returns the correct file extension
00022     public void test1_getFileExtension() {

```



```

00024     File testFile1 = new File("testfile.csv");
00025     // Expects "csv" as the outcome
00026     assertEquals("csv", fileSystem.getFileExtension(testFile1));
00027 }
00028
00029
00030 @Test
00031 // This test checks that getFileExtension() returns the correct file extension
00032 public void test2_getFileExtension(){
00033     File testFile2 = new File("testfile.pdf");
00034     // Expects "pdf" as the outcome
00035     assertEquals("pdf", fileSystem.getFileExtension(testFile2));
00036 }
00037
00038
00039 @Test
00040 // This test checks that getFileExtension() returns the correct file extension
00041 public void test3_getFileExtension(){
00042     File testFile3 = new File("testfiledocs");
00043     // Expects "" as the outcome
00044     assertEquals("", fileSystem.getFileExtension(testFile3));
00045 }
00046
00047
00048 @Test
00049 // This test checks that getFileExtension() returns the correct file extension
00050 public void test4_getFileExtension(){
00051     File testFile4 = new File("");
00052     // Expects "" as the outcome
00053     assertEquals("", fileSystem.getFileExtension(testFile4));
00054 }
00055
00056
00057 @Test
00058 // This test checks that getFileExtension() returns the correct file extension
00059 public void test5_getFileExtension(){
00060     File testFile5 = new File("testfile.csv.csv");
00061     // Expects "csv.csv" as the outcome
00062     assertEquals("csv", fileSystem.getFileExtension(testFile5));
00063 }
00064
00065
00066 @Test
00067 // This test checks that getFileExtension() returns the correct file extension
00068 public void test6_getFileExtension(){
00069     File testFile6 = new File("testfile..");
00070     // Expects "" as the outcome
00071     assertEquals("", fileSystem.getFileExtension(testFile6));
00072 }
00073
00074
00075 @Test
00076 // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
00077 public void test7_checkFileFormat(){
00078     File testFile7 = new File("testfile.csv");
00079     // Expects True as the outcome
00080     assertEquals(true, fileSystem.checkFileFormat(testFile7));
00081 }
00082
00083
00084 @Test
00085 // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
00086 public void test8_checkFileFormat(){
00087     File testFile8 = new File("testfile.pdf");
00088     // Expects False as the outcome
00089     assertEquals(false, fileSystem.checkFileFormat(testFile8));
00090 }
00091
00092
00093 @Test
00094 // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
00095 public void test9_checkFileFormat(){
00096     File testFile9 = new File("testfile..");
00097     // Expects False as the outcome
00098     assertEquals(false, fileSystem.checkFileFormat(testFile9));
00099 }
00100
00101
00102 @Test
00103 // This test checks that checkFileFormat() returns boolean indicating whether file format is
correct
00104 public void test10_checkFileFormat(){
00105     File testFile10 = new File("testfile.csv.csv");
00106     // Expects True as the outcome

```

```

00107         assertEquals(true, fileSystem.checkFileFormat(testFile10));
00108     }
00109
00110
00111     @Test
00112     // This test checks that readFile() reads and returns correct information based on election file
00113     public void test11_readFile() throws IOException{
00114
00115         // Create temporary csv file containing election information
00116         File testFile11 = File.createTempFile("testFile11", ".csv");
00117         FileWriter writer = new FileWriter(testFile11);
00118         writer.write("CPL\n3\nDemocratic, Republican, New Wave\nFoster, Volz, Pike\nGreen, Xu,
Wang\nJacks, Rosen\n3\n8");
00119         writer.close();
00120
00121         // Call the method being tested
00122         fileSystem.readFile(testFile11);
00123
00124         // Stores candidates from all parties
00125         List<List<String>> expectedCandidatesList = Arrays.asList(
00126             // List of candidates from each party
00127             Arrays.asList("Foster, Volz, Pike"),
00128             Arrays.asList("Green, Xu, Wang"),
00129             Arrays.asList("Jacks, Rosen")
00130         );
00131
00132         // Test that election information were read correctly from testFile12
00133         assertEquals("CPL", fileSystem.electionType);
00134         assertEquals(3, fileSystem.numOfCandidates);
00135         assertEquals(Arrays.asList("Democratic", "Republican", "New Wave"), fileSystem.candidates);
00136         assertEquals(expectedCandidatesList, fileSystem.candidatesList);
00137         assertEquals(3, fileSystem.numOfSeats);
00138         assertEquals(8, fileSystem.numOfVotes);
00139     }
00140
00141
00142     @Test
00143     // This test checks that readFile() reads and returns correct information based on election file
00144     public void test12_readFile() throws IOException{
00145
00146         // Create temporary csv file containing election information
00147         File testFile12 = File.createTempFile("testFile12", ".csv");
00148         FileWriter writer = new FileWriter(testFile12);
00149         writer.write("IR\n4\nRosen (D), Kleinberg (R), Chou (I), Royce (L)\n8");
00150         writer.close();
00151
00152         // Call the method being tested
00153         fileSystem.readFile(testFile12);
00154
00155         // Test that election information were read correctly from testFile12
00156         assertEquals("IR", fileSystem.electionType);
00157         assertEquals(4, fileSystem.numOfCandidates);
00158         assertEquals(Arrays.asList("Rosen (D)", "Kleinberg (R)", "Chou (I)", "Royce (L)"),
fileSystem.candidates);
00159         assertEquals(8, fileSystem.numOfVotes);
00160     }
00161
00162
00163     @Test
00164     // This test checks that openFile() checks the file format and then reads file
00165     public void test13_openFile(){
00166         File testFile13 = new File("Project2/testing/CPL_18-3-2023.csv");
00167
00168         // Check if file format is True
00169         fileSystem.openFile(testFile13);
00170         assertTrue(fileSystem.checkFileFormat(testFile13));
00171
00172         // Initialize boolean
00173         boolean functionWasRun = true;
00174         // Check if readFile() is called using try and catch
00175         try { fileSystem.readFile(testFile13); } catch (Exception e) { functionWasRun = true; }
00176         assertTrue(functionWasRun);
00177     }
00178
00179
00180
00181     @Test
00182     // This test checks that openFile() checks the file format and then reads file
00183     public void test14_openFile(){
00184         File testFile14 = new File("Project1/testing/CPL_18-3-2023.pdf");
00185
00186         fileSystem.openFile(testFile14);
00187         assertFalse(fileSystem.checkFileFormat(testFile14));
00188
00189         // Check if ReadFile() function was run
00190         boolean functionWasRun = false;
00191         try { fileSystem.readFile(testFile14); } catch (Exception e) { functionWasRun = false; }

```

```

00192         assertFalse(functionWasRun);
00193     }
00194
00195 }
00196
00197
00198
00199

```

4.7 finalRanking.java

```

00001 import javax.swing.*;
00002 import java.lang.Math;
00003 import java.util.*;
00004
00017 public class finalRanking {
00018
00019     static Map<String, List<Integer> finalRanking = new LinkedHashMap<>();
00020
00025     public finalRanking () {
00026         finalRanking = checkForTie();
00027     }
00028
00034     public Map<String, List<Integer> checkForTie () {
00035         rankings rank = new rankings();
00036         int yesOrNo;
00037         int finalResult;
00038
00039         // If the election type is Closed Party Listing (CPL)
00040         if(fileSystem.electionType.equals("CPL")) {
00041             List<Map.Entry<String, List<Integer>> entries = new
00042             ArrayList<>(rankings.ranking.entrySet());
00043             int totalSeats = fileSystem.numOfSeats;
00044             // check if the last rank from num of seats is tie with the next one
00045             for(int i = totalSeats - 1; i < fileSystem.numOfCandidates; i++) {
00046                 if(entries.get(i).getValue().get(1) == entries.get(i+1).getValue().get(1)) {
00047                     if(entries.get(i+1).getValue().get(1) == entries.get(i+2).getValue().get(1)){
00048                         // if more than 2 have equal value
00049                         yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a pool coin
00050                         toss","Pool Coin Toss",JOptionPane.YES_NO_OPTION);
00051                         if(yesOrNo == JOptionPane.YES_OPTION) {
00052                             // if yes, run a pool coin toss
00053                             finalResult = poolCoinToss(i + 2);
00054                             if(finalResult == 0) {
00055                                 for(int k = 0; k < totalSeats - 1; k++) {
00056                                     finalRanking.put(entries.get(k).getKey(),
00057                                     entries.get(k).getValue());
00058                                 }
00059                                 finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
00060                             }
00061                         }
00062                     }
00063                     else if(finalResult == 1){
00064                         for(int k = 0; k < totalSeats - 1; k++) {
00065                             finalRanking.put(entries.get(k).getKey(),
00066                             entries.get(k).getValue());
00067                         }
00068                         finalRanking.put(entries.get(i+1).getKey(),
00069                         entries.get(i+1).getValue());
00070                     }
00071                     else {
00072                         for(int k = 0; k < totalSeats - 1; k++) {
00073                             finalRanking.put(entries.get(k).getKey(),
00074                             entries.get(k).getValue());
00075                         }
00076                         finalRanking.put(entries.get(i+2).getKey(),
00077                         entries.get(i+2).getValue());
00078                     }
00079                     return finalRanking;
00080                 }
00081             }
00082         }
00083         else {
00084             // if only 2 has equal value
00085             yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a fair coin
00086             toss","Fair Coin Toss",JOptionPane.YES_NO_OPTION);
00087             if(yesOrNo == JOptionPane.YES_OPTION) {
00088                 // if yes, run a fair coin toss.
00089                 finalResult = fairCoinToss();
00090                 if(finalResult == 0){
00091                     for(int k = 0; k < totalSeats - 1; k++) {
00092                         finalRanking.put(entries.get(k).getKey(),
00093                         entries.get(k).getValue());
00094                     }
00095                     finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
00096                 }
00097             }
00098         }
00099     }
00100 }

```

```

00085                     else {
00086                         for(int k = 0; k < totalSeats - 1; k++) {
00087                             finalRanking.put(entries.get(k).getKey(),
entries.get(i+1).getValue());
00088                         }
00089                             finalRanking.put(entries.get(i+1).getKey(),
entries.get(i).getValue());
00090                     }
00091                     return finalRanking;
00092                 }
00093             }
00094             else {
00095                 for(int k = 0; k < totalSeats; k++) {
00096                     finalRanking.put(entries.get(k).getKey(), entries.get(k).getValue());
00097                 }
00098             }
00099             return finalRanking;
00100         }
00101     }
00102
00103     // If the election type is Instant Runoff Voting (IR)
00104     else if(fileSystem.electionType.equals("IR")) {
00105         List<Map.Entry<String, List<Integer>>> entries = new
ArrayList<>(rankings.ranking.entrySet());
00106         int i = 0;
00107         // check if there is a tie between the first place
00108         if(entries.get(i).getValue().get(1) == entries.get(i+1).getValue().get(1)) {
00109             if(entries.get(i+1).getValue().get(1) == entries.get(i+2).getValue().get(1)) {
00110                 // check if more than 2 has tie
00111                 yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a pool coin
toss","Pool Coin Toss",JOptionPane.YES_NO_OPTION);
00112                 if(yesOrNo == JOptionPane.YES_OPTION) {
00113                     // run a pool coin toss if there is a tie
00114                     finalResult = poolCoinToss(i + 2);
00115                     if(finalResult == 0) {
00116                         finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
00117                     }
00118                     else if(finalResult == 1){
00119                         finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
00120                     }
00121                     else {
00122                         finalRanking.put(entries.get(i+2).getKey(), entries.get(i).getValue());
00123                     }
00124                 }
00125             }
00126             else {
00127                 // if there is only 2 has tie
00128                 yesOrNo = JOptionPane.showConfirmDialog(null,"Do you want to run a fair coin
toss","Fair Coin Toss",JOptionPane.YES_NO_OPTION);
00129                 if(yesOrNo == JOptionPane.YES_OPTION) {
00130                     // run a fair coin toss if there is a tie
00131                     finalResult = fairCoinToss();
00132                     if(finalResult == 0) {
00133                         finalRanking.put(entries.get(i).getKey(), entries.get(i).getValue());
00134                     }
00135                     else {
00136                         finalRanking.put(entries.get(i+1).getKey(), entries.get(i).getValue());
00137                     }
00138                 }
00139             }
00140             return finalRanking;
00141         }
00142     }
00143
00144     // If the election type is invalid
00145     else {
00146         JOptionPane.showMessageDialog(null,"The election type is not recognized.,"ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
00147     }
00148
00149     return finalRanking;
00150 }
00151
00152 public int fairCoinToss () {
00153     // If the election type is Closed Party Listing (CPL)
00154     if(fileSystem.electionType.equals("CPL")) {
00155         return (int)(Math.random() * 2) + 0;
00156     }
00157
00158     // If the election type is Instant Runoff Voting (IR)
00159     else if(fileSystem.electionType.equals("IR")) {
00160         return (int)(Math.random() * 2) + 0;
00161     }
00162
00163     // Displays error message if an error is detected
00164     else {
00165         JOptionPane.showMessageDialog(null,"The election type is not recognized.,"ERROR

```

```

MESSAGE", JOptionPane.ERROR_MESSAGE);
00171     }
00172     return 0;
00173 }
00174
00181 public int poolCoinToss (int range) {
00182     // If the election type is Closed Party Listing (CPL)
00183     if(fileSystem.electionType.equals("CPL")) {
00184         return (int)(Math.random() * range) + 0;
00185     }
00186
00187     // If the election type is Instant Runoff Voting (IR)
00188     else if(fileSystem.electionType.equals("IR")) {
00189         return (int)(Math.random() * range) + 0;
00190     }
00191
00192     // Displays error message if an error is detected
00193     else {
00194         JOptionPane.showMessageDialog(null,"The election type is not recognized.", "ERROR
MESSAGE", JOptionPane.ERROR_MESSAGE);
00195     }
00196     return 0;
00197 }
00198 }

```

4.8 finalRankingTest.java

```

00001 import org.junit.Test;
00002 import static org.junit.Assert.*;
00003 import java.util.*;
00004
00019 public class finalRankingTest {
00020
00021     @Test
00022     // This test case checks if checkfortie() performs pool coin toss when election type is CPL
00023     public void test1_checkForTie(){
00024
00025         fileSystem.electionType = "CPL";
00026         // set election type to IR
00027         fileSystem.numOfCandidates = 4;
00028         // set number of parties to 4
00029         fileSystem.numOfSeats = 5;
00030         // set number of seats to 5
00031         fileSystem.numOfVotes = 9;
00032         // set number of votes to 4
00033         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
00034         add parties
00035         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
00036         add 9 arbitrary ballots
00037         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00038         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00039         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00040         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00041         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00042         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00043         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00044         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00045
00046         // create an object of the rankings class and count ballots for each candidate
00047         rankings rank = new rankings();
00048         rank.checkRanking(countBallot.ballotWithName);
00049
00050         // +-----+
00051         // | Results: |
00052         // +-----+
00053         // | "Democratic", (6,3) |
00054         // | "Republican", (6,3) |
00055         // | "Reform", (6,3) |
00056         // | "Green", (9,0) |
00057         // +-----+
00058
00059         // create an object of the finalrankings class to check for tie
00060         finalRanking fr = new finalRanking();
00061
00062         // Check if checkForTie() function was executed
00063         boolean CheckForTieWasRun = true;
00064         try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun = false; }
00065         assertTrue(CheckForTieWasRun );
00066
00067         // Check if poolCoinToss() was run when more than two parties have equal votes
00068         boolean poolCoinTossWasRun = true;
00069         try { fr.poolCoinToss(3); } catch (Exception e) { poolCoinTossWasRun = false; }
00070         // poolCoinToss was run because a tie exist between more than two parties

```

```

00065         assertTrue(poolCoinTossWasRun);
00066
00067         // Check if fairCoinToss() was not run given that poolCoinToss() was run
00068         boolean fairCoinTossWasRun = false;
00069         try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = true; }
00070         // fairCoinToss was not run because a tie exist between more than two parties
00071         assertFalse(fairCoinTossWasRun);
00072     }
00073 }
00074
00075
00076 @Test
00077 // This test case checks if checkfortie() performs fair coin toss when election type is IR
00078 public void test2_checkForTie() {
00079
00080     fileSystem.electionType = "IR"; // set
00081     // election type to IR
00082     fileSystem.numOfCandidates = 3; // set number
00083     // of candidates to 3
00084     fileSystem.numOfVotes = 8; // set number
00085     // of votes to 4
00086     fileSystem.candidates.add("Cedric Tan"); // add first
00087     // candidate
00088     fileSystem.candidates.add("Bryan Lee"); // add second
00089     // candidate
00090     fileSystem.candidates.add("Sherryl Ooi"); // add third
00091     // candidate
00092     fileSystem.ballot.add(Arrays.asList(2, 1, 3)); // add 4 arbitrary
00093     // ballots with rankings
00094     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00095     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00096     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00097     fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00098     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00099     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00100     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00101
00102     // create an object of the rankings class and count ballots for each candidate
00103     rankings rank = new rankings();
00104     rank.checkRanking(countBallot.ballotWithName);
00105
00106     // +-----+
00107     // | Results: |
00108     // +-----+
00109     // | "Cedric", (0,0,2,6) |
00110     // | "Bryan", (0,4,3,1) |
00111     // | "Sherryl", (0,4,3,1) |
00112     // +-----+
00113
00114     // create an object of the finalrankings class to check for tie
00115     finalRanking fr = new finalRanking();
00116
00117     // Check if checkForTie() function was executed
00118     boolean CheckForTieWasRun = true;
00119     try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun = false; }
00120     assertTrue(CheckForTieWasRun);
00121
00122     // Check if poolCoinToss() was not run given that there is only a tie between two parties
00123     boolean poolCoinTossWasRun = false;
00124     try { fr.poolCoinToss(3); } catch (Exception e) { poolCoinTossWasRun = true; }
00125     // poolCoinToss was run because a tie exist between two parties
00126     assertFalse(poolCoinTossWasRun);
00127
00128     // Check if fairCoinToss() was run given that there is only a tie between two parties
00129     boolean fairCoinTossWasRun = true;
00130     try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = false; }
00131     // fairCoinToss was run because a tie exist between two parties
00132     assertTrue(fairCoinTossWasRun);
00133 }
00134
00135 @Test
00136 // This test case checks if checkfortie() does not perform any toss given that there is no tie
00137 // when election type is CPL
00138 public void test3_checkForTie() {
00139
00140     fileSystem.electionType = "CPL";
00141     // set election type to IR
00142     fileSystem.numOfCandidates = 4;
00143     // set number of parties to 4
00144     fileSystem.numOfSeats = 5;
00145     // set number of seats to 5
00146     fileSystem.numOfVotes = 9;
00147     // set number of votes to 4
00148     fileSystem.candidates.addAll(Arrays.asList("Democratic", "Republican", "Reform", "Green")); //
00149     // add parties
00150     fileSystem.ballot.add(Arrays.asList(1,0,0,0)); //

```

```

    add 9 arbitrary ballots
00139     fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00140     fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00141     fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00142     fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00143     fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00144     fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00145     fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00146     fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00147
00148     // create an object of the rankings class and count ballots for each candidate
00149     rankings rank = new rankings();
00150     rank.checkRanking(countBallot.ballotWithName);
00151
00152     // +-----+
00153     // | Results:      |
00154     // +-----+
00155     // | "Democratic", (5,4) |
00156     // | "Republican", (7,2) |
00157     // | "Reform", (6,3)    |
00158     // | "Green", (9,0)     |
00159     // +-----+
00160
00161     // create an object of the finalrankings class to check for tie
00162     finalRanking fr = new finalRanking();
00163
00164     // Test that checkForTie() function was executed
00165     boolean CheckForTieWasRun = true;
00166     try { fr.checkForTie(); } catch (Exception e) { CheckForTieWasRun = false; }
00167     assertTrue(CheckForTieWasRun);
00168
00169     // Test that poolCoinToss() was not run given there is no tie
00170     boolean poolCoinTossWasRun = false;
00171     try { fr.poolCoinToss(3); } catch (Exception e) { poolCoinTossWasRun = true; }
00172     assertFalse(poolCoinTossWasRun);
00173
00174     // Test that fairCoinToss() was not run given there is no tie
00175     boolean fairCoinTossWasRun = false;
00176     try { fr.fairCoinToss(); } catch (Exception e) { fairCoinTossWasRun = true; }
00177     assertFalse(fairCoinTossWasRun);
00178 }
00179
00180
00181 @Test
00182 // This test case checks if checkfortie() does not perform any toss if election type is CPL
00183 public void test4_fairCoinToss(){
00184
00185     fileSystem.electionType = "IR"; // set
00186     election type to IR
00187     fileSystem.numOfCandidates = 3; // set number
00188     of candidates to 3
00189     fileSystem.numOfVotes = 8; // set number
00190     of votes to 4
00191     fileSystem.candidates.add("Cedric Tan"); // add first
00192     candidate
00193     fileSystem.candidates.add("Bryan Lee"); // add second
00194     candidate
00195     fileSystem.candidates.add("Sherryl Ooi"); // add third
00196     candidate
00197     fileSystem.ballot.add(Arrays.asList(2, 1, 3)); // add 4 arbitrary
00198     ballots with rankings
00199     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00200     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00201     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00202     fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00203     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00204     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00205     fileSystem.ballot.add(Arrays.asList(3, 1, 2));
00206
00207     // create an object of the rankings class and count ballots for each candidate
00208     rankings rank = new rankings();
00209     rank.checkRanking(countBallot.ballotWithName);
00210
00211     // +-----+
00212     // | Results:      |
00213     // +-----+
00214     // | "Cedric", (0,0,2,6) |
00215     // | "Bryan", (0,4,3,1)  |
00216     // | "Sherryl", (0,4,3,1) |
00217     // +-----+
00218
00219     // create an object of the finalrankings class to check for tie
00220     finalRanking fr = new finalRanking();
00221
00222     // Test that fairCoinToss returns a value between 0 and 1 indicating the winning party
00223     int result = fr.fairCoinToss();
00224     assertTrue(result >= 0 && result <= 1);

```

```

00218
00219     }
00220
00221     @Test
00222     // This test case checks if checkfortie() does not perform any toss if election type is CPL
00223     public void test5_poolCoinToss(){
00224
00225         fileSystem.electionType = "CPL";
00226         // set election type to IR
00227         fileSystem.numOfCandidates = 4;
00228         // set number of parties to 4
00229         fileSystem.numOfSeats = 5;
00230         // set number of seats to 5
00231         fileSystem.numOfVotes = 9;
00232         // set number of votes to 4
00233         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
00234         add parties
00235         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
00236         add 9 arbitrary ballots
00237         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00238         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00239         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00240         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00241         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00242         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00243         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00244         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00245         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00246
00247         // create an object of the rankings class and count ballots for each candidate
00248         rankings rank = new rankings();
00249         rank.checkRanking(countBallot.ballotWithName());
00250
00251         // +-----+
00252         // | Results: |
00253         // +-----+
00254         // | "Democratic", (6,3) |
00255         // | "Republican", (6,3) |
00256         // | "Reform", (6,3) |
00257         // | "Green", (9,0) |
00258         // +-----+
00259
00260         // create an object of the finalrankings class to check for tie
00261         finalRanking fr = new finalRanking();
00262
00263         // Test that fairCoinToss returns a value between 0 and 1 indicating the winning party
00264         int result = fr.poolCoinToss(3);
00265         assertTrue(result >= 0 && result <= 2);
00266     }
00267 }
00268 }

```

4.9 rankings.java

```

00001 import javax.swing.*;
00002 import java.lang.Math;
00003 import java.util.*;
00004 import java.util.Map.*;
00005
00006 public class rankings {
00007
00008     // A Linked Hash Map to store rankings of each candidate
00009     static Map<String, List<Integer>> ranking = new LinkedHashMap<>();
00010
00011     // An array list to store ??
00012     static List<Map<String, List<Integer>>> displayList = new ArrayList<>();
00013
00014     public rankings () {
00015         countBallot ballotCount = new countBallot();
00016         checkMajority(ballotCount.ballotWithName());
00017     }
00018
00019     public Map<String, List<Integer>> checkRanking (Map<String, List<Integer>> ballotWithName) {
00020
00021         // Check if election type is CPL
00022         if(fileSystem.electionType.equals("CPL")) {
00023             List<Map.Entry<String, List<Integer>>> entryList = new
00024             ArrayList<>(ballotWithName.entrySet());
00025
00026             // Sort the entryList in descending order by second value in each list
00027             Collections.sort(entryList, Comparator.comparingInt((Map.Entry<String, List<Integer>> e) ->
00028             e.getValue().get(1)).reversed());
00029
00030         }
00031     }
00032 }

```



```

00050
00051         // Create a new LinkedHashMap with the sorted entries
00052         for (Map.Entry<String, List<Integer> entry : entryList) {
00053             ranking.put(entry.getKey(), entry.getValue());
00054         }
00055
00056         // Print the sorted map
00057         return ranking;
00058     }
00059
00060     // Check if election type is IR
00061     else if (fileSystem.electionType.equals("IR")) {
00062         List<Map.Entry<String, List<Integer>> entryList = new
ArrayList<> (ballotWithName.entrySet());
00063
00064         // Sort the entryList in descending order by second value in each list
00065         Collections.sort(entryList, Comparator.comparingInt((Map.Entry<String, List<Integer> e) ->
e.getValue().get(1)).reversed());
00066
00067         // Create a new LinkedHashMap with the sorted entries
00068         for (Map.Entry<String, List<Integer> entry : entryList) {
00069             ranking.put(entry.getKey(), entry.getValue());
00070         }
00071
00072         // Print the sorted map
00073         return ranking;
00074     }
00075
00076     // Displays error message if an error is detected
00077     else {
00078         JOptionPane.showMessageDialog(null, "The election type is not recognized.", "ERROR
MESSAGE", JOptionPane.ERROR_MESSAGE);
00079     }
00080     return ranking;
00081 }
00082
00083 public void checkMajority (Map<String, List<Integer> ballotWithName) {
00084
00085     // Checks if election type is CPL
00086     if (fileSystem.electionType.equals("CPL")) {
00087         ranking = checkRanking(ballotWithName);
00088     }
00089
00090     // Checks if election type is IR
00091     else if (fileSystem.electionType.equals("IR")) {
00092         int majority = (int) fileSystem.numOfVotes * 50 / 100;
00093         ranking = checkRanking(ballotWithName);
00094         Map.Entry<String, List<Integer> entry = ranking.entrySet().iterator().next();
00095         int initialRank = 1;
00096
00097         // While there is no majority
00098         while (entry.getValue().get(1) < majority) {
00099
00100             displayList.add(new LinkedHashMap<> (ranking));
00101             List<Entry<String, List<Integer>> entryList = new ArrayList<Map.Entry<String,
List<Integer>>> (ranking.entrySet());
00102             Entry<String, List<Integer> lastEntry = entryList.get(entryList.size() - 1);
00103             List<Integer> subList = lastEntry.getValue();
00104             subList.set(1, 0);
00105             ranking.put(lastEntry.getKey(), subList);
00106             ranking.remove(lastEntry.getKey());
00107
00108             // Redistribute votes
00109             List<List<Integer> checkBallot = new ArrayList<> ();
00110             checkBallot = fileSystem.ballot;
00111             int check = fileSystem.candidates.indexOf(lastEntry.getKey());
00112             for (int i = 0; i < fileSystem.numOfVotes; i++) {
00113                 if (fileSystem.ballot.get(i).get(check) == initialRank) {
00114                     for (int j = 0; j < fileSystem.numOfCandidates; j++) {
00115                         if (fileSystem.ballot.get(i).get(j) == initialRank+1) {
00116                             List<Integer> subBallotList =
ranking.get(fileSystem.candidates.get(j));
00117                             int oldValue = subBallotList.get(1);
00118                             oldValue++;
00119                             subBallotList.set(1, oldValue);
00120                             ranking.put(fileSystem.candidates.get(j), subBallotList);
00121                         }
00122                     }
00123                 }
00124             }
00125             ranking = checkRanking(ranking);
00126             entry = ranking.entrySet().iterator().next();
00127         }
00128         displayList.add(new LinkedHashMap<> (ranking));
00129     }
00130 }
00131
00132 }
00133
00134 }
00135
00136

```

```

00137         // Displays error message if an error is detected
00138     else {
00139         JOptionPane.showMessageDialog(null,"The election type is not recognized.,"ERROR
MESSAGE",JOptionPane.ERROR_MESSAGE);
00140     }
00141 }
00142 }

```

4.10 rankingsTest.java

```

00001 import org.junit.Test;
00002 import static org.junit.Assert.*;
00003 import java.util.*;
00004
00019 public class rankingsTest {
00020
00021     @Test
00022     // This test case checks if checkRanking() sorts each candidate based on votes for CPL election
00023     public void test1_checkRankingsTest() {
00024
00025         fileSystem.electionType = "CPL";
00026         // set election type to IR
00027         fileSystem.numOfCandidates = 4;
00028         // set number of parties to 4
00029         fileSystem.numOfVotes = 7;
00030         // set number of votes to 4
00031         fileSystem.candidates.addAll(Arrays.asList("Democratic","Republican","Reform","Green")); //
add parties
00032         fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
add 7 arbitrary ballots
00033         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00034         fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00035         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00036         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00037         fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00038         fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00039
00040         // expectedRanking is created and used to compare results with actualRanking to determine if
ranking is correct
00041         Map<String, List<Integer> expectedRanking = new LinkedHashMap<>();
00042         expectedRanking.put("Republican", Arrays.asList(3,4));
00043         expectedRanking.put("Democratic", Arrays.asList(5,2));
00044         expectedRanking.put("Reform", Arrays.asList(6,1));
00045         expectedRanking.put("Green", Arrays.asList(7,0));
00046
00047         // Create an object of rankings class and check ranking for all candidates
00048         rankings rank = new rankings();
00049         Map<String, List<Integer> actualRanking = rank.checkRanking(countBallot.ballotWithName);
00050
00051         // Test that the ranking for each candidate has been determined correctly
00052         assertEquals(expectedRanking, actualRanking);
00053     }
00054
00055     @Test
00056     // This test case checks if checkRanking() sorts each candidate based on votes for IR election
00057     public void test2_checkRankingsTest() {
00058
00059         fileSystem.electionType = "IR"; // set
election type to IR
00060         fileSystem.numOfCandidates = 3; // set number
of candidates to 3
00061         fileSystem.numOfVotes = 4; // set number
of votes to 4
00062         fileSystem.candidates.add("Cedric Tan"); // add first
candidate
00063         fileSystem.candidates.add("Bryan Lee"); // add second
candidate
00064         fileSystem.candidates.add("Sherryl Ooi"); // add third
candidate
00065         fileSystem.ballot.add(Arrays.asList(3, 2, 1)); // add 4 arbitrary
ballots with rankings
00066         fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00067         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00068         fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00069
00070         // expectedRanking is created and used to compare results with actualRanking to determine if
ranking is correct
00071         Map<String, List<Integer> expectedRanking = new LinkedHashMap<>();
00072         expectedRanking.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
00073         expectedRanking.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
00074         expectedRanking.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));

```

```

00074
00075 // Create an object of rankings class and check ranking for all candidates
00076 rankings rank = new rankings();
00077 Map<String, List<Integer> actualRanking = rank.checkRanking(countBallot.ballotWithName);
00078
00079 // Test that the ranking for each candidate has been determined correctly
00080 assertEquals(expectedRanking, actualRanking);
00081 }
00082
00083
00084 @Test
00085 // This test case checks if checkRanking() sorts each candidate based on votes for CPL election
00086 public void test3_checkRanking(){
00087
00088     fileSystem.electionType = "CPL";
00089 // set election type to IR
00090     fileSystem.numOfCandidates = 4;
00091 // set number of parties to 4
00092     fileSystem.numOfVotes = 7;
00093 // set number of votes to 4
00094     fileSystem.candidates.addAll(Arrays.asList("Democratic", "Republican", "Reform", "Green")); //
00095 add parties
00096     fileSystem.ballot.add(Arrays.asList(0,1,0,0)); //
00097 add 7 arbitrary ballots
00098     fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00099     fileSystem.ballot.add(Arrays.asList(0,0,1,0));
00100     fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00101     fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00102     fileSystem.ballot.add(Arrays.asList(1,0,0,0));
00103     fileSystem.ballot.add(Arrays.asList(0,1,0,0));
00104
00105 // expectedResultsAfterCheckingMajority is created and used to compare with actual results
00106 after checking for Majority
00107     Map<String, List<Integer> expectedRanking = new LinkedHashMap<>();
00108     expectedRanking.put("Republican", Arrays.asList(3,4));
00109     expectedRanking.put("Democratic", Arrays.asList(5,2));
00110     expectedRanking.put("Reform", Arrays.asList(6,1));
00111     expectedRanking.put("Green", Arrays.asList(7,0));
00112
00113 // Create an object of rankings class and check ranking for all candidates
00114 rankings rank = new rankings();
00115 Map<String, List<Integer> actualRanking = rank.checkRanking(countBallot.ballotWithName);
00116
00117 // Test that the ranking for each candidate has been determined correctly after checking for
00118 Majority
00119 assertEquals(expectedRanking, actualRanking);
00120 }
00121
00122 @Test
00123 // This test case checks if checkMajority() finds a majority among the candidates for IR election
00124 public void test4_checkMajority(){
00125
00126     fileSystem.electionType = "IR"; // set
00127 election type to IR
00128     fileSystem.numOfCandidates = 3; // set number
00129 of candidates to 3
00130     fileSystem.numOfVotes = 9; // set number
00131 of votes to 9
00132     fileSystem.candidates.add("Cedric Tan"); // add first
00133 candidate
00134     fileSystem.candidates.add("Bryan Lee"); // add second
00135 candidate
00136     fileSystem.candidates.add("Sherryl Ooi"); // add third
00137 candidate
00138     fileSystem.ballot.add(Arrays.asList(3, 2, 1)); // add 9 arbitrary
00139 ballots with rankings
00140     fileSystem.ballot.add(Arrays.asList(2, 1, 3));
00141     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00142     fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00143     fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00144     fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00145     fileSystem.ballot.add(Arrays.asList(2, 1, 3));
00146     fileSystem.ballot.add(Arrays.asList(2, 1, 3));
00147     fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00148
00149 // expectedResultsAfterCheckingMajority is created and used to compare with actual results
00150 after checking for Majority
00151     Map<String, List<Integer> expectedResultsAfterCheckingMajority = new LinkedHashMap<>();
00152     expectedResultsAfterCheckingMajority.put("Sherryl Ooi", Arrays.asList(0, 0, 3, 3));
00153     expectedResultsAfterCheckingMajority.put("Bryan Lee", Arrays.asList(0, 6, 3, 3));
00154     expectedResultsAfterCheckingMajority.put("Cedric Tan", Arrays.asList(0, 0, 3, 3));
00155
00156 // Create an object of rankings class and check for majority
00157 rankings rank = new rankings();
00158 rank.checkMajority(countBallot.ballotWithName);
00159
00160 // Test that the ranking for each candidate has been determined correctly after checking for

```

```

Majority
00146         assertEquals(expectedResultsAfterCheckingMajority, countBallot.ballotWithName);
00147     }
00148
00149     @Test
00150     // This test case checks if checkMajority() finds a majority among the candidates for IR election
00151     public void test5_checkMajority(){
00152
00153         fileSystem.electionType = "IR";                                     // set
election type to IR
00154         fileSystem.numOfCandidates = 3;                                   // set number
of candidates to 3
00155         fileSystem.numOfVotes = 4;                                       // set number
of votes to 4
00156         fileSystem.candidates.add("Cedric Tan");                         // add first
candidate
00157         fileSystem.candidates.add("Bryan Lee");                         // add second
candidate
00158         fileSystem.candidates.add("Sherryl Ooi");                       // add third
candidate
00159         fileSystem.ballot.add(Arrays.asList(3, 2, 1));                   // add 4 arbitrary
ballots with rankings
00160         fileSystem.ballot.add(Arrays.asList(2, 3, 1));
00161         fileSystem.ballot.add(Arrays.asList(3, 2, 1));
00162         fileSystem.ballot.add(Arrays.asList(1, 3, 2));
00163
00164         // expectedResultsAfterCheckingMajority is created and used to compare with actual results
after checking for Majority
00165         Map<String, List<Integer> expectedResultsAfterCheckingMajority = new LinkedHashMap<>();
00166         expectedResultsAfterCheckingMajority.put("Sherryl Ooi", Arrays.asList(0, 3, 1, 0));
00167         expectedResultsAfterCheckingMajority.put("Bryan Lee", Arrays.asList(0, 0, 2, 2));
00168         expectedResultsAfterCheckingMajority.put("Cedric Tan", Arrays.asList(0, 1, 1, 2));
00169
00170         // Create an object of rankings class and check for majority
00171         rankings rank = new rankings();
00172         rank.checkMajority(countBallot.ballotWithName);
00173
00174         // Test that the ranking for each candidate has been determined correctly after checking for
Majority
00175         assertEquals(expectedResultsAfterCheckingMajority, countBallot.ballotWithName);
00176     }
00177 }

```

4.11 votingSystem.java

```

00001 public class votingSystem {
00002     public static void main (String args[]) {
00003         displayResults results = new displayResults();
00004     }
00005 }

```

Index

- checkFileFormat
 - fileSystem, 11
- checkForTie
 - finalRanking, 17
- checkMajority
 - rankings, 20
- checkRanking
 - rankings, 21
- countBallot, 5
 - countBallot, 5
- countBallotTest, 6
 - test1_countBallot, 6
 - test2_countBallot, 7
 - test3_countBallot, 7
- displayResults, 7
 - displayResults, 8
 - generateAuditFile, 8
 - showResults, 8
- displayResultsTest, 8
 - test1_generateAuditFile, 9
 - test2_generateAuditFile, 9
 - test3_showResults, 9
 - test4_showResults, 9
- fairCoinToss
 - finalRanking, 17
- fileSystem, 10
 - checkFileFormat, 11
 - fileSystem, 10
 - fileSystemRead, 11
 - getFileExtension, 11
 - openFile, 12
 - readFile, 12
- fileSystemRead
 - fileSystem, 11
- fileSystemTest, 13
 - test10_checkFileFormat, 13
 - test11_readFile, 13
 - test12_readFile, 14
 - test13_openFile, 14
 - test14_openFile, 14
 - test1_getFileExtension, 14
 - test2_getFileExtension, 14
 - test3_getFileExtension, 14
 - test4_getFileExtension, 15
 - test5_getFileExtension, 15
 - test6_getFileExtension, 15
 - test7_checkFileFormat, 15
 - test8_checkFileFormat, 15
 - test9_checkFileFormat, 15
- finalRanking, 16
 - checkForTie, 17
 - fairCoinToss, 17
 - finalRanking, 16
 - poolCoinToss, 17
- finalRankingTest, 18
 - test1_checkForTie, 18
 - test2_checkForTie, 18
 - test3_checkForTie, 19
 - test4_fairCoinToss, 19
 - test5_poolCoinToss, 19
- generateAuditFile
 - displayResults, 8
- getFileExtension
 - fileSystem, 11
- main
 - votingSystem, 23
- openFile
 - fileSystem, 12
- poolCoinToss
 - finalRanking, 17
- rankings, 19
 - checkMajority, 20
 - checkRanking, 21
 - rankings, 20
- rankingsTest, 21
 - test1_checkRankingsTest, 22
 - test2_checkRankingsTest, 22
 - test3_checkRanking, 22
 - test4_checkMajority, 22
 - test5_checkMajority, 22
- readFile
 - fileSystem, 12
- showResults
 - displayResults, 8
- src/countBallot.java, 25
- src/countBallotTest.java, 26
- src/displayResults.java, 27
- src/displayResultsTest.java, 29
- src/fileSystem.java, 32
- src/fileSystemTest.java, 34
- src/finalRanking.java, 37
- src/finalRankingTest.java, 39
- src/rankings.java, 42

- src/rankingsTest.java, 44
- src/votingSystem.java, 46
- test10_checkFileFormat
 - fileSystemTest, 13
- test11_readFile
 - fileSystemTest, 13
- test12_readFile
 - fileSystemTest, 14
- test13_openFile
 - fileSystemTest, 14
- test14_openFile
 - fileSystemTest, 14
- test1_checkForTie
 - finalRankingTest, 18
- test1_checkRankingsTest
 - rankingsTest, 22
- test1_countBallot
 - countBallotTest, 6
- test1_generateAuditFile
 - displayResultsTest, 9
- test1_getFileExtension
 - fileSystemTest, 14
- test2_checkForTie
 - finalRankingTest, 18
- test2_checkRankingsTest
 - rankingsTest, 22
- test2_countBallot
 - countBallotTest, 7
- test2_generateAuditFile
 - displayResultsTest, 9
- test2_getFileExtension
 - fileSystemTest, 14
- test3_checkForTie
 - finalRankingTest, 19
- test3_checkRanking
 - rankingsTest, 22
- test3_countBallot
 - countBallotTest, 7
- test3_getFileExtension
 - fileSystemTest, 14
- test3_showResults
 - displayResultsTest, 9
- test4_checkMajority
 - rankingsTest, 22
- test4_fairCoinToss
 - finalRankingTest, 19
- test4_getFileExtension
 - fileSystemTest, 15
- test4_showResults
 - displayResultsTest, 9
- test5_checkMajority
 - rankingsTest, 22
- test5_getFileExtension
 - fileSystemTest, 15
- test5_poolCoinToss
 - finalRankingTest, 19
- test6_getFileExtension
 - fileSystemTest, 15
- test7_checkFileFormat
 - fileSystemTest, 15
- test8_checkFileFormat
 - fileSystemTest, 15
- test9_checkFileFormat
 - fileSystemTest, 15
- votingSystem, 23
 - main, 23