README (Just for testing, not our official README):

For our unit testing, we often test more than one method within the same test class, as having a different test class for every single method seems rather wasteful, and would likely result in this document being absurdly long given the length of each table (it's already 90 pages, which should be sufficiently long). The methods that are tested by each testing class are indicated in the test description, as well as under the heading that asks us to indicate which methods/functions are being used.

Unit Testing:

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: testGettersAndSetters() **Name(s) of Testers: Jack Soderwall**

Test Description: We tested to see if our setters properly set data fields as the election was run via testing of the getter methods associated with an OPL election. All public getter methods are tested via this manner.

Automated: yes_X_ no ___

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **The constructor (and therefore determineWinner()), getParticipatingParties(),**

	getCandidates(), getNumBallots(), getSeats()
Results: Pass __X__ Fail_____	

Preconditions for Test: **We initialized and ran an OPL election using one of our test files, which would set the variables as needed.**	

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Check if participatingParties.size() is equal to 3.	The participatingParties list for this OPL election run, as retrieved by getParticipatingParties()	3	3	Test passed
2	Check if candidates.size() is equal to 4.	The list of candidates for this OPL election, as retrieved by getCandidates()	4	4	Test passed

3	Check if numBallots is equal to 6	The numBallots for this election, as retrieved by getNumBallots( )	6	6	Test passed
4	Check if seats is equal to 2	The seats for this OPL election, as retrieved by getSeats()	2	2	Test passed
5	Get the party names of the 3 parties involved in this election	The parties for this OPL election, as retrieved by getPartyName()	R, D, and I in that order, given how the test was run	R, D, I	Test passed

Post condition(s) for Test:

A media and audit file will have been produced in the same directory as this code with the details form this OPL election.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: tieBreakTest() **Name(s) of Testers: Jack Soderwall**

Test Description: Tests if there is a bias in our tiebreak system by making sure that no candidate wins by more than 10 fair coin tosses, or relatively close to 50% odds.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **TiebreakTest.java**
  - **We use a separate, default OPL constructor in order to make it more "tester friendly" for this, as well as the tieBreak() method from Election.java**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized an OPL object and two candidates to test our tiebreaks on.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	Break 100 ties in a row, counting up the number of wins for each candidate.	The tieBreak() method will return the winner of between the two candidates. Their win totals will be incremented.	No results at this stage.	No results at this stage.	Just the initial setup step.
2	Make sure that the absolute value of the difference between the two win totals is less than 20.	The win totals for the two candidates after running 100 tiebreaks between them.	Absolute value of difference should be less than 20.	After several runs, it appears that our tiebreak is unbiased, as the test has always passed. Any failures would likely be an anomaly.	Considering this test passed.

Post condition(s) for Test:

Since we used a default constructor that does not run the entire system, which is the purpose of this new constructor, there are no side effects to running this test.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: testName() **Name(s) of Testers: Jack Soderwall**

Test Description: Simply tests the creation of a candidate via the constructor, the creation of a party via the constructor, and the ability to use the getName() getter method.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **candidateTest.java**
  - **We test the Party and Candidate constructors as well as the getName() method from the candidate class.**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized a candidate and a party to test our methods on, using our constructors of course.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check if our candidates name is properly retrieved by the getName() method	The name of our created candidate, as retrieved by the getName() method	“Kanye West”	“Kanye West”	Test passed

Post condition(s) for Test:

None really, there are no side effects to this test except for the creation of a candidate named Kanye West.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testBallotPreferences() **Name(s) of Testers: Sean Carter**

Test Description: Tests if ballot preferences are able to be retrieved and manipulated properly. Also tests getPreferredCandidate and eliminatePreferredCandidate and their associated getters.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **BallotTest.java**
  - **We use the constructor, getPreferredCandiate, and eliminatePreferredCandidate**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized a list of candidates in order to test our various methods on.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check if preferred candidate for ballot1 is equal to ballot3	ballot1.getPreferredCandidate().getName()	They should not be equal	They are not equal	This part passes
2	Check that preferred candidate for ballot1 is Joe Biden	ballot1.getPreferredCandidate.getName()	Should be Joe Biden	It is Joe Biden	Test passes
3	Eliminate preferred candidate from ballot 1 and then check if the new preferred candidate is Donald Trump	ballot1.eliminatePreferredCandidate(), ballot1.getPreferredCandidate.getName())	The new preferred candidate should be Donald Trump	It should be Donald Trump	Test passes
4	Get the names of the eliminated preferred	ballot2.eliminatePreferredCandidate().getName()	Should get Donald Trump, Kanye West, then	It gets them in the correct order	Test passes

	candidates from ballot2		Alexander Hamilton		
5	Eliminates ballot3's one preferred candidate, and then makes sure that getPreferredCandidate is null	ballot3.eliminatePreferredCandidate.getName(), ballot3.getPreferredCandidate()	Should get Kanye West and then null after these operations are performed in the order specified to the left.	It does get Kanye West and then null as expected	Test passes

Post condition(s) for Test:

Ballot 3 should have no preferred candidate left, as well as the effects of the manipulations detailed above. Overall, since this is just a small test file, the system itself shouldn't be affected.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testBallotLine() **Name(s) of Testers: Sean Carter**

Test Description: Tests the storing and retrieval of ballot data

BallotTest.java

- **testBallotLine()**
- **Ballot()**
- **getBallotInfo()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized a list of candidates in order to test our various methods on. Also, we initialized 3 ballot objects with ballot info passed in to test.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that ballot1's ballotInfo has stored the correct string	String "1,2,3,4" and "1,2,3,4" passed into the ballot constructor to initialize ballotInfo	Should be equal to one another "1,2,3,4"	They are equal "1,2,3,4"	Test Passed

2	Check that ballot2's ballot info has stored the correct string	String "4,1,3,2" and "4,1,3,2" passed into the ballot constructor to initialize ballotInfo	Should be equal to one another "4,1,3,2"	They are equal "4,1,3,2"	Test Passed
3	Check that ballot2's ballotInfo stored the correct string	String ",,,1" and ",,,1" passed into the ballot constructor to initialize ballotInfo	Should be equal to one another ",,,1"	They are equal ",,,1"	Test Passed

Post condition(s) for Test:

ballot1, ballot2 and ballot3 will all have the correct balllotInfo stored after the lines are passed to the constructor.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testVoteCount() **Name(s) of Testers: Conor Brown**

Test Description: Tests if getVoteCount() and if increment vote count will work.

CandidateTest.java

- **testVoteCount()**
- **Candidate()**
- **Party()**
- **getVoteCount()**
- **incrementVoteCount()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

Initialize party in order to initialize candidates. Then initialize a candidate to run tests upon.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Confirm that candidate has correct initial vote count	0 is what the initial vote count should be and compared to kanyeWest.get VoteCount()	Should be equal to 0	The values are equal 0	Test Passed
2	Increment candidate vote count with incrementVoteC	IncrementVote Count() increments kanyeWests's	Should be equal to 1	The values are equal 1	Test Passed

	ount and test that getVoteCount returns the correct incremented count	votes to 1 and kanyeWest.get VoteCount()			
3	Increment candidate vote count again and ensure that it sets the vote count correctly	Increment kayneWest's vote count again and compare it to kanyeWest.get VoteCount()	Should be equal to 2	The values are equal 2	Test Passed
4	Call incrementVoteC ount with parameter that will add the passed in value to the candidates current vote count	Call kanyeWest.incr ementVoteCou nt(2018) then compared kanyeWest.get VoteCount() to 2020	Should be equal 2020	The values are equal 2020	Test Passed

Post condition(s) for Test:

kayneWest's vote count will be equal to 2020 and kanyeWest.getVoteCount() returns 2020.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testCompareTo() **Name(s) of Testers: Conor Brown**

Test Description: Test the testCompareTo() method to ensure it correctly compares two candidates

- **CandidateTest.java**
  - **testCompareTo()**
  - **Candidate()**
  - **Party()**
  - **compareTo()**
  - **incrementVoteCount()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized a party to initialize two candidates to compare to each other.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	Increment candidate kanyeWest's vote count and compare it to candidate northWests	Will check to see if -1 is equal to kanyeWest.compareTo(nothWest) because kanyeWest has a greater number of votes	The values should be equal -1	The values are equal -1	Test Passed
2	Increment candidate nothWests's vote count to 1 and then compare to kanyeWestVote count	Check to see when kanyeWest.compareTo(northWest) returns 0 meaning their vote count is equal	The values should be equal 0	The values are equal 0	Test Passed
3	Increment northWest's vote count again and compare to kanyeWest's	kanyeWest.compareTo(northWest) should return 1 now that northWest's vote count is greater than kanyeWest's	The values should be equal 1	The values are equal 1	Test Passed

Post condition(s) for Test:

northWest's vote count will be equal to 2 and kanyeWest's vote count will be equal to 1.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ Test Date: 03/13/21

Test Case ID#: testSetParty

Name(s) of Testers: Conor Brown

Test Description: Test the setParty() function in Candidate class

- **CandidateTest.java**
  - **testSetParty()**
  - **Party()**
  - **Candidate()**
  - **getPart()**
  - **setParty()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

Initialize two parties patriots and buccaneers to test that the constructor sets te correct party when passed in and then tests that setParty() correctly changes that party on the candidate tomBrady.

Step	Test Step	Test	Expected	Actual	

#	Description	Data	Result	Result	Notes
1	Check to see that the constructor correct assign tomBrady's party at patriots	tomBrady.getParty() is compared to patriots the Party object	Should be equal. patriots	The values are equal patriots	Test Passed
2	Change tomBrady's party to buccaneers and use tomBrady.getParty to see if the party has been updated	tomBrady.getParty() is compared to buccaneers the party	The values should be equal buccaneers	The values are equal buccaneers	Test Passed

Post condition(s) for Test:

Candidate tomBrady's party will be equal to buccaneers.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testInitValues() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Party class constructor assigns the correct initial values.

- **PartyTest.java**
  - **getTotalPartyVotes()**
  - **getPartyMembers()**
  - **getSeatsAllocated()**
  - **getRemainder()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

Create a Party presidential to create an object of class Party with name "Presidential".

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check to see if the correct party name has been initialized by calling presidential.getPartyName()	"Presidential" is compared to presidential.getPartyName()	The values should be equal. "Presidential"	The values are equal. "Presidential"	Test passed

2	Check to see if the Party's total vote count has been initialized correctly. Call presidential.getTotalPartyVotes()	Initial party vote count should be equal to 0. presidential.getTotalPartyVotes() is compared to 0.	The values should be equal. 0	The values are equal. 0	Test passed
3	Check if affiliatedPartyMembers has been initialized but with no Candidates within it yet. Call presidential.getPartyMembers().size()	The size of partyMembers should initially be equal to 0. presidential.getPartyMembers().size() should be equal to 0	The values should be equal. 0	The values are equal. 0	Test passed
4	Check to see that seatsAllocated is initialized to 0. Call presidential.getSeatsAllocated().	presidential.getSeatsAllocated() should return 0 after initialization, compare this to 0	The values should be equal. 0	The values are equal. 0	Test passed
5	Check to see that presidential's remainder is equal to 0. Call presidential.getRemainder()	presidential.getRemainder() should return 0. Compare that to 0	The values should be equal. 0	The values are equal. 0	Tests passed

Post condition(s) for Test:

A Party presidential will be initialized with all the correct values after calling the constructor.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/13/21**

Test Case ID#: testPartMembers() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Party class's affiliatedPartyMembers works as intended.

- **PartyTest.java**
  - **Party()**
  - **Candidate()**
  - **addCandidate()**
  - **getPartyMemebers()**
  - **incrementVoteCount()**
  - **sortPartyMembersByVote()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

Create a Party presidential to create an object of class Party with name "Presidential". Create four Candidate objects that are affiliated with Party presidential.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Test if the size of affiliatedPartyMembers for Party presidential is equal to 4 after 4 candidates have been created with the presidential Party affiliation	4 and getPartyMemebers().size()	The values should be equal. 4	The values are equal. 4	Test passed
2	Test that the affiliatedPartyMembers are in the correct initial position.	Compares the Candidate abeLicoln to presidential.getPartyMembers().get(3) which will return the Candidate at position 3 in	The values should be equal. abeLincoln	The values are equal abeLincoln	Test passed
3	Test that sorPartyMembersByVotes() works as intended	Check that Candidate teddyRoosevelt is now in last with the least amount of votes. PartyMembers().get(3) should now return teddyRoosevelt	The values should be equal. teddyRoosevelt	The values are equal. teddyRoosevelt	Test passed

Post condition(s) for Test:

Party presidential will have 4 affiliated Candidates and the affiliatedPartyMembers list will be sorted by each candidate's number of votes.

OPL System Tests:

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: OPLTest1() **Name(s) of Testers: Jack Soderwall**

Test Description: Runs an Open Party Listing election on a file called OPLTest1.csv. This test contained a tie, so we tested our ability to break ties.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **Tests our running of the OPL election on a small file, thus testing all of the functionality of the OpenPartyListingElection() class of our program**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized an OpenPartyListingElection class using a scanner and then ran an OPL election. We are then checking if the results are what we expect.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that 2 candidates are seated	opl.getSeatedCandidates() aka a list of the candidates that were seated in the election	2	2	Test passed
2	Check that the first candidate in the seated candidates list is Billy and he is of party D	seatedCandidates.get(0).getName(), seatedCandidates.get(0).getParty().getPartyName()	Should be Billy and D if ran in the order specified on the left	Billy, D	Test passed
3	Check that the two parties get allocated 1 seat	getParty().getSeatsAllocated()	1 and 1 since we perform 1 check for each of the	1, 1	Test passed

			parties in this election		
4	Check that the number of seated candidates is equal to the number of seats up for grabs at the start of the election	opl.getSeats()	True	True	Test passed

Post condition(s) for Test:

Audit and media files will be placed into the directory that the code was run from that details all of the steps in the given election. Relevant results will be displayed to the terminal.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: OPLTest2() **Name(s) of Testers: Jack Soderwall**

Test Description: Runs an Open Party Listing election on a file called OPLTest2.csv

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **Tests our running of the OPL election on a small file, thus testing all of the functionality of the OpenPartyListingElection() class of our program**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized an OpenPartyListingElection class using a scanner and then ran an OPL election. We are then checking if the results are what we expect.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that 3 candidates are seated	opl.getSeatedCandidates() aka a list of the candidates that were seated in the election	3	3	Test passed

2	Check that Pike's party got 5 total votes	seatedCandidates.get(0).getParty().getTotalPartyVote()	5	5	Test passed
3	Check that the seated candidate at position 0 is Pike	getName() aka the name of the seated candidate at position 0	Pike	Pike	Test passed
4	Check that the party of the seated candidate at index 1 has 3 votes	seatedCandidates.get(1).getParty().getTotalPartyVote()	3	3	Test passed
5	Check that Jones and Foster are the seated candidates at indices 1 and 2	getName() aka the names of the candidates	Jones, Foster	Jones, Foster	Test passed

Post condition(s) for Test:

Audit and media files will be placed into the directory that the code was run from that details all of the steps in the given election.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: OPLTestSpecial() **Name(s) of Testers: Jack Soderwall**

Test Description: Runs an Open Party Listing election on a file called OPLTestSpecial.csv. This is a very special case where a party wins more seats than it has candidates, so we tested our ability to handle that.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **Tests our running of the OPL election on a small file, thus testing all of the functionality of the OpenPartyListingElection() class of our program**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized an OpenPartyListingElection class using a scanner and then ran an OPL election. We are then checking if the results are what we expect.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that 3 candidates are seated	opl.getSeatedCandidates() aka a list of the candidates that were seated in the election	3	3	Test passed
2	Check that the seated candidate at index 0 got 4 votes	getTotalPartyVote aka the total number of votes cast to the party	4	4	Test passed
3	Check that the seated candidate at position 0 is Sean	getName() aka the name of the seated candidate at position 0	Sean	Sean	Test passed
4	Check that the parties of the candidates seated at indices 1 and 2 each have 1 vote	getTotalPartyVote() AKA the vote total for the party these	1	1	Test passed

		candidates are in			
5	Check that the number of seated candidates equals the number of seats up for grabs at the start of the election	getSeats aka retrieving the number of seats that were open at the start of this election	True	True	Test passed

Post condition(s) for Test:

Audit and media files will be placed into the directory that the code was run from that details all of the steps in the given election.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: testLargeOPLFile() **Name(s) of Testers: Jack Soderwall**

Test Description: Tests the functionality of our OPL Election on a file with 100,000 ballots. Uses the file "hundredk_ballots_opl.csv"

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **Tests our running of the OPL election on a very large file, thus testing all of the functionality of the OpenPartyListingElection() class of our program**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

We initialized an OpenPartyListingElection class using a scanner and then ran an OPL election. We are then checking if the results are what we expect.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Check that 3 candidates are seated	opl.getSeatedCandidates() aka a list of the candidates that were seated in the election	3	3	Test passed

2	Check that the seated candidate at index 0 got 99,999 votes	getTotalPartyVote aka the total number of votes cast to the party	99,999	99,999	Test passed
3	Check that the seated candidate at position 0 is named Pike and his party is D	getName() aka the name of the seated candidate at position 0 and getParty() aka the party of the seated candidate at position 0	Pike, D	Pike, D	Test passed
4	Check that the seated candidate at position 1 is Foster and his party is D	getName() aka the name of the seated candidate at position 1 and getParty() aka the party of the seated candidate at position 1	Foster	D	Test passed
5	Check that the seated candidate at position 1 is Smith and his party is I	getName() aka the name of the seated candidate at position 2 and getParty() aka the party of the seated candidate at position 2	Smith	I	Test passed

Post condition(s) for Test:

Audit and media files will be placed into the directory that the code was run from that details all of the steps in the given election.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/24/21**

Test Case ID#: OPLTest3Way() **Name(s) of Testers: Jack Soderwall**

Test Description: Tests the ability for the program to handle 3 way/multi-way ties with grace. I ran the program multiple times and manually inspected the audit files to make sure the ties were being broken as expected, and that two different candidates were being seated in this test case.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **We use the OPLElection constructor, and by default the determineWinner() method**

Automated: yes__ no _X_

Results: Pass __X__ Fail_____

Preconditions for Test:

A file called OPL3Way.csv existed and was available for testing.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize and run and OPL election.	The audit file	Results not checked yet	Results not checked yet	N/A
2	Check audit files to make sure that two candidates won a seat from the correct party, and that they are unique.	The audit file produced by the program.	Two winners, each from party R, out of a pool of either Jack, Sean, or Joe	Through multiple runs, there were always multiple winners and all winners came from the pool of desired candidates.	Manual inspection passed

Post condition(s) for Test:

Various audit and media files were created for the multiple runs of this program that I did.

System Testing for Instant Runoff:

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: testFixedFile() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Instant Runoff Election works when a candidate has an initial clear majority.

- **IRElectionTest.java**
  - **InstantRunoffElection()**
  - **getElectionWinner()**
  - **getPartyName()**
  - **getVoteCount()**
  - **getParty()**
  - **getPartyMemebers()**
  - **getParticipatingParties()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

A specified ballot file has been created to test this scenario. The tests opens the file and a scanner to read it. An InstantRunoffElection object is created to run the election and the winner is stored.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Confirm the correct winner was found	Compare “Watters” to winner.getName()	Should be equal. Watters	They are equal Watters	Test passed
2	Confirm the winner has the correct party name	Compare “P” to winner.getParty().getaArtyName()	The values should be equal. “P”	The values are equal “P”	Test Passed
3	Confirm the correct number of votes was found for the winner.	Compare 3, the expect votes, to winner.getVoteCount()	The values should be equal 3	The values are equal. 3	Test passed
4	Ensure that all members were assigned to the correct party	Compare 4, the number of Candidates in the winner’s party to winner.getParty.	The values should be equal. 4	The values are equal. 4	Test Passed

		getPartyMembers.size()			
5	Ensure that the number of participatingParties is correct	Compare 1, the expect number of participating parties with ir.getParticipatingParties.size()	The values should be equal. 1	The values are equal 1	Test passed

Post condition(s) for Test:

An IR election will have run with the specified file and a winner declared.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: testComeback() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Instant Runoff Election works when a comeback to win after runoff.

Automated: yes_X_ no ___

- **IRElectionTest.java**
  - **InstantRunoffElection()**
  - **getElectionWinner()**
  - **getParty()**
  - **getPartyName()**

- **getVoteCount()**
- **getElectionWinner()**

Results: Pass __X__ Fail_____

Preconditions for Test:

A specified ballot file has been created to test this scenario. The test opens the file and a scanner to read it. An InstantRunoffElection object is created to run the election and the winner is stored.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Tests that the correct winner is found.	“Kleinberg”, the expected winner, is compared to winner.getName()	The values should be equal. “Kleinberg”	The values are equal. “Kleinberg”	Test Passed
2	Confirm that the winner is assigned the correct party	“R”, the expected party name of the winner, is compared to	The values should be equal. “R”	The values are equal. “R”	Test Passes

		winner.getParty. getPartyName()			
3	Confirm that the winner has the correct number of votes	4, the expected number of votes for winner, is compared to winner.getVote Count	The values should be equal. 4	The values are equal. 4	Test Passes

Post condition(s) for Test:

An IR election will have run with the specified file and a winner declared.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: testFourWayTie() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Instant Runoff Election works when there is a four way tie.

Automated: yes_X_no ___

- **IRElectionTest.java**
  - **InstantRunoffElection()**

- **getEliminatedCandidates()**
- **getParticipatingParties()**
- **getNumBallots()**
- **getCandidates()**

Results: Pass __X__ Fail_____

Preconditions for Test:

A specified ballot file has been created to test this scenario. The test opens the file and a scanner to read it. An InstantRunoffElection object is created to run the election.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Ensure that there are 3 eliminated Candidates	Compare 3, the expect number of eliminated Candidates to ir.getEliminatedCandidates.getsize() 3	The values should be equal.	The values are equal. 3	Test Passed
2	Ensure that the correct number of participating	Compare 4, the expected number of	The values should be equal.	The values are equal.	Test Passes

	parties was found and stored	participating parties to ir.getParticipatingParties().size()	4	4	
3	Ensure that only one ballot remains following the election due to tie breaking and eliminations	Compare 1, the expected number of remaining ballots, to ir.getNumBallots()	The values should be equal. 1	The values are equal. 1	Test Passes
4	Ensure that only one candidate remains following the election due to tie breaking and eliminations	Compare 1, the expected number of remaining ballots to ir.getCandidates.size()	The values should be equal. 1	The values are equal 1	Test Passes

Post condition(s) for Test:

An IR election will have run with the specified file and a winner declared.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#:
testCandidatesWithLastNames() **Name(s) of Testers: Conor Brown**

Test Description: Test that the Instant Runoff Election works when there is a four way tie.

- **IRElectionTest.java**
  - **InstantRunoffElection()**
  - **getName()**
  - **getParty()**
  - **getPartyName()**

Automated: yes_X_no ___

Results: Pass ____ Fail____X____

Preconditions for Test:

A specified ballot file has been created to test this scenario. The test opens the file and a scanner to read it. An InstantRunoffElection object is created to run the election.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Checks if the name of the	winner.getName() or the name	The values should be equal.	The values are not equal. The space is	Test Fails

	winner is equal to “Mike Douglas”	of the winner of the IR election		missing in between Mike and Douglas. This is addressed in the bug list.	
2	Checks if the winner is of the party D.	winner.getParty().getPartyName() or the name of the party of the election winner	The values should be equal.	The values are equal.	Test Fails due to the above test not passing, but the values will be equal.

Post condition(s) for Test:

An IR election will have run with the specified file and a winner declared.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _X_ **Test Date: 03/13/21**

Test Case ID#: fileNotFound() **Name(s) of Testers: Conor Brown**

Test Description: Test that the proper exception is thrown when an incorrect file path is given to ElectionDriver.

- **IRElectionTest.java**
  - **ElectionDriver.main()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

Enter in a non-existent file path which will throw an exception when passed to ElectionDriver.main().

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Check to see if ElectionDriver.main() will throw the correct exception when provided with an incorrect file path	Compare an illegalArgument Exception with what exception ElectionDriver.main() throws when passed an incorrect file name	The values should be equal. IllegalArgumentException	The values are equal. IllegalArgumentException	Test Passed

Post condition(s) for Test:

The specified path will not be found and an exception should be thrown.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _x_ **Test Date: 03/13/21**

Test Case ID#: testIRWriteFunctions() **Name(s) of Testers: Conor Brown**

Test Description: Test that the output functions displayResultsToTerminal(), writeMediaFile(), writeAuditFileHeader() and writeToAuditFile() work correctly for an Instant runoff election.

- **OutputTest.java**
  - **InstantRunoffElection()**

Automated: yes__ no _x__

Results: Pass __X__ Fail_____

Preconditions for Test:

A file with the correct file path will be given to run an instant runoff election to observe the produced output from various output functions.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run an instant runoff election and inspect the output sent to the terminal.	The output sent to the terminal by writeToTerminal()	Expected Result below.	The correct output was produce, equal to expected	Test passed
2	Inspect the media file produced by the run election	The file created by writeMediaFile()	Expected Result below.	The correct output was produce, equal to expected	Test passed
3	Inspect the audit file produced by the run election	The file created by various calls to writeToAuditFile() and the call to writeAuditFileHeader()	Expected Result below.	The correct output was produce, equal to expected	Test passed

Post condition(s) for Test:

The various outputs will be produced. Note: the file names and time when run will change based on runtime. But at the time this was run the output was correct. Additionally, winners and tiebreaks could have different outcomes producing slightly different output.

Terminal expected output

Program completed successfully

Election Summary

Election Type: Instant Runoff

Total Ballots Counted: 5

Winner: Watters Votes: 3, 60.0% of the vote

Elimination order:

An audit file with the name AuditFile-2021-03-25-16-11-28.txt has been produced in C:\csci5801\repo-Team21\Project1

A media file with the name MediaFile-2021-03-25-16-11-28.txt has been produced in C:\csci5801\repo-Team21\Project1

Media file expected output

2021-03-25-16-11-28 Instant Runoff Election Results

Total Votes Cast: 5

Candidates (4)

-Watters: P

Elimination Order

Final votes garnered by each remaining candidate

-Watters Votes: 3

-Challou Votes: 1

-Wrenn Votes: 1

-Weissman Votes: 0

Winner: Watters Votes: 3, 60.0% of the vote

Audit file expected output

Election Type: Instant Runoff

Number of ballots: 5

Number of candidates: 4

Candidates:

-Watters, Party: P

-Weissman, Party: P

-Challou, Party: P

-Wrenn, Party: P

Initial ballot awarding:

1,,2 awarded to Watters

1,2,4,3 awarded to Watters

,,1, awarded to Challou

1,2,, awarded to Watters

,,,1 awarded to Wrenn

Initial votes per candidate:

-Watters Votes: 3

-Weissman Votes: 0

-Challou Votes: 1

-Wrenn Votes: 1

Majority found, Winner declared:

Final Vote Count:

Winner: Watters Votes: 3, 60.0% of the vote

Runner up(s):

-Challou Votes: 1

-Wrenn Votes: 1

-Weissman Votes: 0

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit __ System _x_ **Test Date: 03/13/21**

Test Case ID#: testOPLWriteFunctions() **Name(s) of Testers: Conor Brown**

Test Description: Test that the output functions displayResultsToTerminal(), writeMediaFile(), writeAuditFileHeader() and writeToAuditFile() work correctly for an open party listing election.

- **OutputTest.java**
  - **OpenPartyListingElection()**

Automated: yes__ no _x__

Results: Pass __X__ Fail_____

Preconditions for Test:

Enter in a file with the correct file path to run an open party listing election to observe the produced output from various output functions.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run an instant runoff election and inspect the output sent to the terminal.	The output sent to the terminal by writeToTerminal()	Expected Result below.	The correct output was produce, equal to expected	Test passed
2	Inspect the media file produced by the run election	The file created by writeMediaFile()	Expected Result below.	The correct output was produce, equal to expected	Test passed
3	Inspect the audit file produced by the run election	The file created by various calls to writeToAuditFile() and the call to	Expected Result below.	The correct output was produce, equal to expected	Test passed

		writeAuditFileHeader()			

Post condition(s) for Test:

The various outputs will be produced. Note: the file names and time when run will change based on runtime. But at the time this was run the output was correct. Additionally, winners and tiebreaks could have different outcomes producing slightly different output.

Terminal expected output

Program completed successfully

Election Summary

Election Type: Open Party Listing

Total Ballots Counted: 6

Participating parties:

-R

-D

-I

Seats allocated: 2

Winners:

-Dan: R, with 1 votes

-Billy: D, with 3 votes

An audit file with the name AuditFile-2021-03-25-17-08-47.txt has been produced in C:\csci5801\repo-Team21\Project1

A media file with the name MediaFile-2021-03-25-17-08-47.txt has been produced in C:\csci5801\repo-Team21\Project1

Media file expected output

2021-03-25-17-08-47 Open Party Listing Election Election Results

---

Total votes cast: 6

Candidates (4) and votes by party:

R, Total Party Votes: 2

-Dan Votes: 1

-Bob Votes: 1

D, Total Party Votes: 3

-Billy Votes: 3

I, Total Party Votes: 1

-Jane Votes: 1

Awarded Seats:

-Dan: R, with 1 votes

-Billy: D, with 3 votes

Audit file expected output

Election Type: Open Party Listing

Number of ballots: 6

Open Seats: 2

Quota: 3 votes

Candidate order as appears on ballots:

-Dan

-Bob

-Billy

-Jane

Candidates by party:

R

-Dan

-Bob

D

-Billy

I

-Jane

Ballots being awarded to each candidate:

,1,, awarded to Bob

,,1, awarded to Billy

1,,, awarded to Dan

,,1, awarded to Billy

,,1, awarded to Billy

,,,1 awarded to Jane

Votes summary for party and each candidate:

R, Total Party Votes: 2

-Dan Votes: 1

-Bob Votes: 1

D, Total Party Votes: 3

-Billy Votes: 3

I, Total Party Votes: 1

-Jane Votes: 1

Initial Seats Awarded:

R: 0 with 2 votes remaining

D: 1 with 0 votes remaining

I: 0 with 1 votes remaining

Updated Seats Awarded after remainder distribution:

R: 1 seat(s)

D: 1 seat(s)

I: 0 seat(s)

Tie between: Dan and Bob

Tie broken: Bob eliminated

Seat winners:

-Dan: R, with 1 votes

-Billy: D, with 3 votes

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: readBallotFileTest() **Name(s) of Testers: Sean Carter**

Test Description: Tests to ensure that readBallotFile() correctly parses the file and stores the correct data on each candidate

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **IRElectionUnitTest.java**
  - **The test calls the default constructor of InstantRunoffElection and calls readBallotFile()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test:

An InstantRunoffElection object must be instantiated and contain a valid ballotFile Scanner object that is not null. The Scanner must have also read the first line of the file.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Verifies that the initial values of the candidate data structures are empty or 0 in value.	The candidates and participatingParties lists for the election	The candidates list and the participating parties must be empty.	Both lists are empty as expected.	Initial setup
2	Calls readBallotFile with a Scanner object (comebackIR.csv) in this instance.	numBallots, candidates/participatingParites lists	The number of ballots should be equal to the actual number cast, and the candidates/participatingParties lists should contain each individual candidate/party.	numBallots and the lists are the correct values and each individual candidate in the candidates list has the correct data attached.	The test passes.

Post condition(s) for Test:

An InstantRunoffElection object has been made containing the correct values respective to what was provided in the ballotFile.

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testSetCandidatesEmpty() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of empty candidate lines.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	Feed an empty string to Election.setCandidates()	Size of Election.candidates	Size is 0	Size is 0	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesIRStandard() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of typical IR candidate lines.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on typical IR candidate line.	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ Test Date: 03/26/21

Test Case ID#: testSetCandidatesIRSingle() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of IR candidate lines containing one candidate.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	Call Election.setCandidates() on IR candidate line with a single candidate.	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testSetCandidatesIRFullNames() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of IR candidate lines containing candidate's full names.

Automated: yes_X_no ___

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**

- **Election.setCandidates()**

Results: Pass ____ Fail____X____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on IR candidate lines containing candidates' full names	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	Candidate names are concatenated; party names are stored as expected	This test fails once again due to the bug with spacing, which has been documented.

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesIRPartyFullNames() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of IR candidate lines containing unabbreviated party names.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass ____ Fail____X____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on IR lines containing unabbreviated party names	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	Party names are concatenated; candidates are stored as expected	One of the tests fails due to spaces in the party name, a well documented bug here. Proceed with caution.

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesIRAbsurdSpacing() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of IR candidate lines containing arbitrary whitespace.

Automated: yes_X_no ___

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**

- **Election.setCandidates()**

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on an IR line containing arbitrary whitespace	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesOPLStandard() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of typical OPL candidate lines.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on a typical OPL candidate line	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testSetCandidatesOPLSingle() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of OPL candidate lines containing a single candidate.

Automated: yes_X_no ___

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on an OPL candidate line containing exactly one entry	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesOPLFullNames() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of OPL candidate lines containing candidates' full names.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass ____ Fail____X____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on an OPL candidate line containing candidates' full names	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	Candidate names are concatenated then stored; parties stored as expected	Once again, this test fails due to the same issues with handling spaces in full names or full party names.

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesAndParties() **Name(s) of Testers: Joe Cassidy**

Test Description: Sanity check on setCandidatesAndParties(). Said function is nothing more than setCandidates() and consolidateParties() called in sequence.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidatesAndParties()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidatesAndParties() on a typical candidate line	Resulting contents of Election.candidates and Election.participating parties	Contents correspond precisely to those in the candidate line.	Contents correspond precisely to those in the candidate line.	
2	Call Election.setCandidatesAndParties	Resulting contents of Election.candidates and Election.participating parties	Contents are empty.	Contents are empty.	

	() on an empty string.				

Post condition(s) for Test: Election.candidates and Election.participatingParties are instantiated with 0 and 0 members, respectively. All candidates and parties contained therein are properly initialized from the given candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#:
testSetCandidatesOPLAbsurdSpacing() **Name(s) of Testers: Joe Cassidy**

Test Description: Check handling of OPL candidate lines containing arbitrary whitespace.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.setCandidates()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call Election.setCandidates() on an OPL candidate line containing arbitrary whitespace	Compare names and parties in Election.candidates to names and parties in line	All names and parties are stored in order of appearance	All names and parties are stored in order of appearance	

Post condition(s) for Test: Election.candidates is instantiated with exactly those candidates given by the candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testConsolidateParties4C4P() **Name(s) of Testers: Joe Cassidy**

Test Description: Check parsing of 4 candidates into 4 parties.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.consolidateParties()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set Election.candidates to 4 candidates in 4 parties, then call	Resulting Election.candidates and Election.participatingParties	Election.participatingParties contains exactly one copy of every party among the candidates.	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the	

	Election.consolidateParties()		candidates. Said parties contain the candidates, and the candidates contain said parties.	candidates, and the candidates contain said parties.	

Post condition(s) for Test: Election.candidates and Election.participatingParties are instantiated with 4 and 4 members, respectively. All candidates and parties contained therein are properly initialized from the given candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testConsolidateParties3C2P() **Name(s) of Testers: Joe Cassidy**

Test Description: Check parsing of 3 candidates into 2 parties.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.consolidateParties()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set Election.candidates to 3 candidates in 2 parties, then call Election.consolidateParties()	Resulting Election.candidates and Election.participatingParties	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates contain said parties.	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates contain said parties.	

Post condition(s) for Test: Election.candidates and Election.participatingParties are instantiated with 3 and 2 members, respectively. All candidates and parties contained therein are properly initialized from the given candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testConsolidateParties2C1P() **Name(s) of Testers: Joe Cassidy**

Test Description: Check parsing of 2 candidates into 1 party.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.consolidateParties()**

Automated: yes_X_no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set Election.candidates to 2 candidates in 1 parties, then call Election.consolidateParties()	Resulting Election.candidates and Election.participatingParties	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates contain said parties.	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates contain said parties.	

Post condition(s) for Test: Election.candidates and Election.participatingParties are instantiated with 2 and 1 members, respectively. All candidates and parties contained therein are properly initialized from the given candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testConsolidateParties0C0P() **Name(s) of Testers: Joe Cassidy**

Test Description: Check parsing of 0 candidates into 0 parties.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **InputHelperTest.java**
  - **Election.consolidateParties()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set Election.candidates to 0 candidates in 0 parties, then call Election.consolidateParties()	Resulting Election.candidates and Election.participatingParties	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates	Election.participatingParties contains exactly one copy of every party among the candidates. Said parties contain the candidates, and the candidates contain said parties.	

			contain said parties.		

Post condition(s) for Test: Election.candidates and Election.participatingParties are instantiated with 0 and 0 members, respectively. All candidates and parties contained therein are properly initialized from the given candidate line.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/26/21**

Test Case ID#: testReadBallotFile() **Name(s) of Testers: Joe Cassidy**

Test Description: Full run of OPL's readBallotFile() and subsequent check of the many variables it must set

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **OPLTest.java**
  - **Election.initializeParameters()**
  - **OpenPartyListingElection.readBallotFile()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: none

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Open a ballot file. Skip the first line then initialize all OPL parameters and call OpenPartyListingElection.readBallotFile().	OpenPartyListingElection.candidates, OpenPartyListingElection.participatingParties, OpenPartyListingElection.numCandidates, OpenPartyListingElection.numBallots	Candidates and parties are stored exactly as they appear in the file. Each candidate and party receives the number of votes allotted to them by the file.	Candidates and parties are stored exactly as they appear in the file. Each candidate and party receives the number of votes allotted to them by the file.	

Post condition(s) for Test:

All data from file "election.files/OPLTest1.csv" is read into the OPLTest object.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: getWinnerTest() **Name(s) of Testers: Sean Carter**

Test Description: Ensures that the IR election is correctly picking the correct winner.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **IRElectionUnitTest.java**
  - **InstantRunoffElection.getWinner()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: The InstantRunoffElection object contains a non-empty list of candidates.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set each candidate in the candidates list to a value for testing with one guaranteed winner and numBallots being set to a very low number. Call getWinner	The candidates list contains all of the participating candidates, totalVoteCount, each candidate's respective vote count, and numBallots, the number of ballots in play.	Whoever has the most votes will be at the front of the list and win via majority.	The person with the most votes wins as intended.	Works accordingly.
2	Increase the number of the votes for some other candidate to be the most and increase numBallots such that can be no majority.	numBallots, the number of ballots in play and totalVoteCount, each candidate's respective vote counts.	The values for these data objects are correctly updated.	The values are correctly updated.	Works as intended.

3	Call getWinner again	Same as before	The candidate who now has the most votes after the vote count updates should win.	The candidate with the most votes is chosen correctly.	Works as intended.

Post condition(s) for Test: The candidates list is ordered by vote and the winner of the election is returned by the function.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: existsMajorityTest() **Name(s) of Testers: Sean Carter**

Test Description: Ensures that the IR election is correctly identifying when a majority is present.

Automated: yes_X_no ___

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **IRElectionUnitTest.java**

- **existsMajority()**

Results: Pass __X__ Fail_____

Preconditions for Test: The InstantRunoffElection object's numBallots value is set to a value relating to the total number of votes in play.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set the candidates list to only contain one candidate with an arbitrary vote count. Then call existsMajority.	The candidates list contains all of the participating candidates, and totalVoteCount, each candidate's respective vote count	existsMajority will return true since a majority is present	The program correctly identifies that a majority is present.	Works accordingly.

2	Add another candidate to the list with the same number of votes as the previous candidate and numBallots is updated accordingly. Then call existsMajority again.	numBallots, the number of ballots in play and totalVoteCount, each candidate's respective vote counts.	existsMajority will return false since no majority is present (candidates are tied).	The program correctly decides there is no majority present.	Works as intended.
3	Update one of the candidates votes to be greater than the other candidate and has a majority in regards to numBallots	Same as before	The program should identify that there is a majority present.	The program correctly identifies that there is a majority present.	Works as intended.

Post condition(s) for Test: The value returned by the function is true if there was a majority and false otherwise.

Project Name: Project 1: Voting System **Team# 21**

Test Stage: Unit _X_ System __ **Test Date: 03/24/21**

Test Case ID#: getLastPlaceCandidatesTest() **Name(s) of Testers: Sean Carter**

Test Description: Ensures that the IR election is correctly obtaining the candidates tied for last place.

- **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**
  - **IRElectionUnitTest.java**
  - **getLastPlaceCandidates()**

Automated: yes_X_ no ___

Results: Pass __X__ Fail_____

Preconditions for Test: The InstantRunoffElection object's candidates list must be non-empty.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set the candidates list to contain two candidates who	The candidates list contains all of the participating candidates, and	The function will return a list containing the two	The program correctly provides the list with the desired candidates.	Works accordingly.

	are tied for last place.	totalVoteCount, each candidate's respective vote count	candidates tied for last.		

Post condition(s) for Test: The value returned by the function is the list of all candidates who are currently tied for last place

PROJECT PORTION 2 NEW TESTS:

For the purposes of this document, the PBI pertaining to our bug from the previous iteration will be dubbed "Whitespace PBI", the PBI pertaining to multiple file handling will be "Multi-file PBI", the PBI pertaining to ballot invalidation will be "Invalidation PBI", and the PBI pertaining to PO election file handling will be "PO PBI". All of this is done for the convenience of both the reader and the creators of this document, and should be rather self explanatory.

WHITESPACE PBI TESTS: NOTE: These are previous tests that did not pass that now pass. Their previous iterations may also be found on this document.

PBI with testing number: PBI: Whitespace PBI, Test ID: testSetCandidatesIRFullNames(), Located in InputHelperTest.java
Team Member(s) responsible: Jack Soderwall and Joe Cassidy
Inputs: A string meant to resemble a line from a ballot file which would contain the candidates and parties for the election.
Tests: 1. Test that candidate names are set properly when they contain spaces, as a first and last name would 2. Test that candidate parties are set properly
Outputs: There are no outputs for this test, as it is checked via assert statements due to this test pertaining to internal data structures that are utilized throughout the execution of a given election.
Passed
Date: 04/28/21

PBI with testing number: PBI: Whitespace PBI, Test ID: testSetCandidatesIRPartyFullNames(), Located in InputHelperTest.java
Team Member(s) responsible: Jack Soderwall and Joe Cassidy

Inputs: A string meant to mimic a line from an election file containing candidate and party information
Tests: 1. Test that candidate names are set properly 2. Test that party names are set properly when they contain spaces
Outputs: There are no outputs for this test, as the results are checked via assert statements due to this test pertaining to internal data structures that are utilized throughout the execution of an election.
Passed
Date: 04/28/21

PBI with testing number: PBI: Whitespace PBI, Test ID: testSetCandidatesOPLFullNames(), Located in InputHelperTest.java
Team Member(s) responsible: Jack Soderwall and Joe Cassidy
Inputs: A string meant to mimic a line from an election file containing candidate and party information
Tests: 1. Test that candidate names are set properly when they contain spaces, as a first and last name would 2. Test that candidate parties are set properly
Outputs: There are no outputs for this test, as the results are checked via assert statements due to this test pertaining to internal data structures that are utilized throughout the execution of an election.
Passed
Date: 04/28/21

PBI with testing number: PBI: Whitespace PBI, Test ID: testSetCandidatesOPLPartyFullNames(), Located in InputHelperTest.java
Team Member(s) responsible: Jack Soderwall and Joe Cassidy
Inputs: A string meant to mimic a line from an election file containing candidate and party information
Tests: 1. Test that candidate names are set properly 2. Test that party names are set properly when they contain spaces
Outputs: There are no outputs for this test, as the results are checked via assert statements due to this test pertaining to internal data structures that are utilized throughout the execution of an election.

Passed
Date: 04/28/21

MULTI-FILE PBI TESTS:

PBI with testing number: Multi-File PBI, Test ID: OPLSpecialMulti(), Located in OPLTest.java
Team Member(s) responsible: Jack Soderwall
Inputs: Two files, each containing roughly one half of this OPL election (OPLSpecial1.csv and OPLSpecial2.csv)
Tests: 1. Checks for the proper total party vote for the first winner 2. Checks for the proper name of the first winner 3. Checks for the proper total party vote for the second winner 4. Checks for the total proper party vote for the third winner 5. Checks that the total number of seated candidates is correct
Outputs: Election stats are displayed to the terminal in accordance with project specifications. An audit file is also produced within the project directory.
Passed (Audit file was also deemed correct via manual inspection)
Date: 04/26/21

PBI with testing number: Multi-File PBI, Test ID: OPLMulti1(), Located in OPLTest.java
Team Member(s) responsible: Jack Soderwall
Inputs: Two files, each containing a portion of this OPL election (OPLMultiOne.csv and OPLMultiTwo.csv)
Tests: 1. Tests that the proper number of seats are allocated 2. Checks for the correct name of the candidate with the most votes 3. Make sure that the candidate with the most votes has been assigned the correct party 4. Makes sure that the two winning parties each were each only allocated one seat
Outputs: Election stats are displayed to the terminal in accordance with project specifications. An audit file is also produced within the project directory.
Passed (Audit file was also deemed correct via manual inspection)
Date: 4/26/21

PBI with testing number: Multi-File PBI, Test ID: OPLMulti2(), Located in OPLTest.java
Team Member(s) responsible: Jack Soderwall
Inputs: Three files, each containing a portion of this OPL election (OPLMulti21.csv, OPLMulti22.csv, and OPLMulti23.csv)
Tests: 1. Check that the proper number of candidates are seated 2. Check that the total party votes of the party with the most votes is correct 3. Check that the name of the person with the most votes is correct 4. Check that the party of the second seated candidate is correct 4. Check that the name of the second seated candidate is correct 5. Check that the name of the third seated candidate is correct. All of these should function properly with the important addition that there are now 3 files at play.
Outputs: Election stats are displayed to the terminal in accordance with project specifications. An audit file is also produced within the project directory.
Passed (Audit file was also deemed correct via manual inspection)
Date: 4/26/21

PBI with testing number: Multi-File PBI, Test ID: OPL3WayMulti(), Located in OPLTest.java
Team Member(s) responsible: Jack Soderwall
Inputs: Two files, each containing a portion of this OPL election (OPL3Way1.csv and OPL3Way2.csv)
Tests: 1. Checks to see if the proper number of candidates are seated 2. Manual inspection of the audit files determines whether or not the 3 way tie is broken in a sufficient manner (in accordance with project specifications)
Outputs: Election stats are displayed to the terminal in accordance with project specifications. An audit file is also produced within the project directory.
Passed (Audit file was also deemed correct via manual inspection)
Date: 4/26/2021

PBI with testing number: Multi-File PBI, Test ID: testNoFile(), Located in MultipleFilesSystemTests.java
Team Member(s) responsible: Jack Soderwall and Conor Brown
Inputs: A string array of arguments that should be empty. This will be passed as the argument to the main class of the program.

Tests: 1. Test that the system exits with the proper message when no file is given
Outputs: System should output a message saying that no file was provided
Passed via manual inspection
Date: 04/28/21

PBI with testing number: Multi-File PBI, Test ID: testOneFile(), Located in MultipleFilesSystemTests.java
Team Member(s) responsible: Jack Soderwall and Conor Brown
Inputs: A string array of arguments that contain only one file path, “/election.files/OPLMultiOne.csv”
Tests: 1. Test that the system runs the election and finds the correct winner.
Outputs: System should output results from the election and produce audit and media files
Passed via manual inspection
Date: 04/28/21

PBI with testing number: Multi-File PBI, Test ID: testMultFilesOPL(), Located in MultipleFilesSystemTests.java
Team Member(s) responsible: Jack Soderwall and Conor Brown
Inputs: A string array of arguments that contain two file paths, “/election.files/OPLMultiOne.csv” and “/election.files/OPLMultiTwo.csv”
Tests: 1. Test that the system runs the election and finds the correct winner with multiple files for OPL.
Outputs: System should output correct results from the election and produce audit and media files
Passed via manual inspection
Date: 04/28/21

PBI with testing number: Multi-File PBI, Test ID: testMultFilesIR(), Located in MultipleFilesSystemTests.java
Team Member(s) responsible: Jack Soderwall and Conor Brown

Inputs: A string array of arguments that contain two file paths, both “/election.files/comebackIR.csv”
Tests: 1. Test that the system runs the election and finds the correct winner with multiple files for IR.
Outputs: System should output correct results from the election and produce audit and media files
Passed via manual inspection
Date: 04/28/21

PBI with testing number: Multi-File PBI, Test ID: testFixedFileMulti(), Located in IRElectionTest.java
Team Member(s) responsible: Sean Carter
Inputs: A list of two scanner objects both pertaining to the same file
Tests: Tests to ensure that the correct winner, number of votes, and number of parties are being presented.
Outputs: The correct results of the election and the consequential audit/media files
Passed via manual inspection
Date: 05/01/21

PBI with testing number: Multi-File PBI, Test ID: testComebackMulti(), Located in IRElectionTest.java
Team Member(s) responsible: Sean Carter
Inputs: A list of two scanner objects both pertaining to the same file
Tests: Tests to ensure that the majority check is properly implemented; if there was no majority in the previous test with one file, then doubling the same data should produce the same outcome.
Outputs: The correct winner of the election and their corresponding vote counts and the audit/media files
Passed via manual inspection
Date: 05/01/21

PBI with testing number: Multi-File PBI, Test ID: testFourWayTieMulti(), Located in IRElectionTest.java
Team Member(s) responsible: Sean Carter
Inputs: A list of two scanner objects both pertaining to the same file
Tests: Tests to ensure that in the tie functionality is preserved between having multiple files present.
Outputs: A random winner for the election and an audit/media file showing the elimination process
Passed via manual inspection
Date: 05/01/21

INVALIDATION PBI TESTS:

PBI with testing number: Invalidation PBI, Test ID: someValidsTest(), Located in IRElectionTest.java
Team Member(s) responsible: Sean Carter
Inputs: A csv file called “IRSomeValids.csv”
Tests: Tests to ensure that ballot invalidation is correctly occurring and the valid ballots are not being invalidated.
Outputs: The correct winner of the election and the corresponding audit/media file.
Passed via manual inspection
Date: 05/01/21

PBI with testing number: Invalidation PBI, Test ID: oneValidTest(), Located in IRElectionTest.java
Team Member(s) responsible: Sean Carter
Inputs: A csv file called “IROneValid.csv”
Tests: Tests to ensure that even though someone has a majority of the votes through invalid ballots, they do not win since they are invalid
Outputs: The correct winner of the election and the corresponding audit/media file.

Passed via manual inspection
Date: 05/01/21

PO PBI TESTS:

PBI with testing number: PO PBI, Test ID: tesPOFile1(), Located in POElectionTest.java
Team Member(s) responsible: Conor Brown
Inputs: One file, POFile1.csv
Tests: Checks if the correct information is stored in from the passed in file. Specifically, number of ballots, number of candidates, and each candidate's name, party name and number of votes
Outputs: None
Passed
Date: 4/28/2021

PBI with testing number: PO PBI, Test ID: tesPOFile2(), Located in POElectionTest.java
Team Member(s) responsible: Conor Brown
Inputs: One file, POFile2.csv
Tests: Checks if the correct information is stored in from the passed in file. Specifically, number of ballots, number of candidates, and each candidate's name, party name and number of votes
Outputs: None
Passed
Date: 4/28/2021

PBI with testing number: PO PBI, Test ID: testPO2Files(), Located in POElectionTest.java
Team Member(s) responsible: Conor Brown
Inputs: Two files, both POFile2.csv
Tests: Checks if the correct information is stored when two files are counted in a PO election.

Specifically, number of ballots, number of candidates, and each candidate's name, party name and number of votes
Outputs: None
Passed
Date: 4/28/2021

PBI with testing number: PO PBI, Test ID: testPO3Files(), Located in POElectionTest.java
Team Member(s) responsible: Conor Brown
Inputs: Three files, all POFile2.csv
Tests: Checks if the correct information is stored when three files are counted in a PO election. Specifically, number of ballots, number of candidates, and each candidate's name, party name and number of votes
Outputs: None
Passed
Date: 4/28/2021