Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __ **Test Date: 2024.04.15**

Test Case ID#: Candidate_Test1 **Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun**

Test Description:

This test case verifies the functionality of the Candidate class, including vote increment, name search, party search, and vote count search.
The tests are stored in CandidateTest.java. Methods being tested include addVote(), getName(), getParty(), and getVotes().

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

Automated: yes__✓_ no ___

Results: Pass __✓___ Fail_____

Preconditions for Test:

1. **The Candidate class and Party class must be correctly implemented.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a Candidate object with initial value	Name: "John Smith", Party: "Test Party"	Candidate object created with specified name and party		
2	Test the getName() method		The name "John Smith" is returned	The name "John Smith" is returnedCorrectly returned the name "John Smith"	
3	Test the getParty() method		The Party object with name "Test Party" is returned	Correctly returned the name "Test Party"	
4	Test the getVotes() method before any votes are added		getVotes() returns 0 before any vote addition	getVotes() correctly returns 0	
5	Test the addVote() method by adding a single vote		getVotes() returns 1 after one vote addition	getVotes() correctly returns 1	
6	Test the addVote() method by adding multiple votes		getVotes() correctly returns the total votes added	After adding 2 more votes, getVotes() returns 3	
7	Test if the candidate has any seats before allocation		hasSeats() returns false before any seat allocation	hasSeats() correctly returns false	
8	Test the allocateSeat() method and check seats		hasSeats() returns true and getSeats() returns 1	hasSeats() returns true; getSeats() returns 1	

9	Test adding multiple seats to the candidate		getSeats() correctly reflects added seats	After adding one more seat, getSeats() returns 2	

Post condition(s) for Test:
The state of the Candidate object reflects the number of votes and seats added during the test. No changes to the Party object. The system remains stable and no exceptions are thrown.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __

Test Date: 2024.04.17

Test Case ID#: Party_Test1

Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun

Test Description:
This test case examines the functionality of the Party class, specifically testing the methods addCandidate(), addVote(), setInitialSeats(), addSecondAllocationSeats(), setRemainderVotes(), and calculateTotalSeats(). The tests verify the management of candidates, vote tallying, seat allocation, and remainder vote handling. Tests are implemented in PartyTest.java.

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

Automated: yes ✓ no

Results: Pass ✓ Fail

Preconditions for Test:
1. **The Party class must be correctly implemented and able to interact with the Candidate class for adding and managing candidates.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a Party object with a name	Name: "Test Party"	Party object created with specified name		
2	Add a candidate to the party's list	Candidate: "Candidate 1"	Party list updates with new candidate	Party contains the added candidate	
3	Add a vote to the party		Party's vote count increments by 1	Party's votes is 1	
4	Set initial seats and verify	Initial Seats: 5	Party's initial seats set to 5	Initial seats are 5	
5	Add second allocation seat and verify		Party's second allocation seats increase by 1	Second allocation seats is 1	
6	Manage and verify remainder votes	Remainder Votes: 10	Party's remainder votes set to 10	Remainder votes are 10	
7	Verify total seats calculation	Initial: 2, Second Allocation: 1	Total seats should be 3	Total seats is 3	Initial and second allocation seats summed
8	Set and reset the won seat flag in the current round	Set to true, then reset	Party has won seat is true, then reset to false	Initially true, then false	Check toggling of flag functionality

Post condition(s) for Test:
The state of the Party object correctly reflects the added candidates, votes, and seats as tested. The system remains stable and no exceptions are thrown.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __ **Test Date: 2024.04.19**

Test Case ID#: ElectionManager_Test1
Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun

Test Description:
This test case evaluates the ElectionManager class's functionality for managing various election types including CPL (Closed Party List), OPL (Open Party List), MPO (Multi-Party Open), and MV (Multi-Vote).
The class is responsible for loading ballot data, conducting elections based on the data, finalizing results, and exporting detailed audit reports.
Specific methods tested include loadBallotData(),

conductElection(), finalizeElectionResults(), and exportAuditFile().

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

Automated: yes ✓ no

Results: Pass __✓___ Fail_____

Preconditions for Test:

1. **Ballot data files must be correctly formatted and accessible. The ElectionManager must be initialized without any pre-loaded election data.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Load ballot data for a CPL election and conduct the election	File: CPL_Voting_1.csv	CPL election data is loaded and election is conducted properly	CPL election data was successfully loaded and election was conducted properly	Verify if parties and votes are correctly loaded and calculated
2	Verify finalization of election results		Correct party is declared winner or ties are resolved	Correct party was declared winner and ties were resolved	Check final results and winner determination logic
3	Export audit file	File: audit_CPL_results.txt	Audit file is created with detailed results	Successfully created audit file with detailed results.	Check final results and winner determination logic
4	Load ballot data for an OPL election and conduct the election	File: OPL_Voting_1.csv	OPL election data is loaded and election is conducted properly	OPL election data was successfully loaded and election was conducted properly	Check if candidate votes are correctly tallied
5	Test handling of multiple file inputs for an MPO election	Files: MPO_Voting_1.csv, MPO_Voting_2.csv	Multiple MPO files are processed correctly	Multiple MPO files were processed correctly	Ensure seamless processing of multiple data sources
6	Load non-existent file and check error handling	File: nonexistent_file.csv	Error is handled gracefully without crashing the system	Error was handled gracefully without crashing the system	Verify robustness of error handling

Post condition(s) for Test:

The ElectionManager object's state accurately reflects the processed election data. All parties, candidates, and votes are correctly tallied, and audit files are generated as expected.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __

Test Date: 2024.04.20

Test Case ID#: ClosedListElection_Test1

Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun

Test Description:

This test case verifies the functionality of the ClosedListElection class, focusing on conducting elections, calculating and allocating seats, and resolving ties through random selection.
Methods tested include conductElection(), calculateSeats(), and the tie-resolution process within calculateSeats() using random selection.
The tests ensure that the class accurately allocates seats to parties based on votes and handles edge cases such as ties.

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

Automated: yes ✓ no

Results: Pass __✓___ Fail_____

Preconditions for Test:

1. **The ClosedListElection class must be properly initialized with a valid number of total seats and votes. All parties participating must be registered within the system.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Conduct an election with no votes	Total Seats: 5, Total Votes: 0	No seats should be allocated	No seats were allocated	This checks the handling of no votes
2	Conduct an election with votes leading to seat allocation	Total Seats: 5, Total Votes: 100, Votes Data: "1"	Proper allocation of seats based on votes	Successfully allocated seats based on votes	Ensure the logic for vote counting and seat allocation is correct
3	Calculate seats based on distributed votes	Total Seats: 5, Party Votes: 20	Parties receive seats proportional to votes	Parties received seats proportional to votes	Check the proportional allocation logic
4	Resolve a tie in seat allocation	Total Seats: 5, Votes: 50-50	Seats are distributed evenly or by	Seats were distributed evenly or by random	Verify the random

	using random selection	split between two parties	random selection	selection	tie-resolution mechanism
5	Test error handling for negative total seats	Total Seats: -1, Total Votes: 100	IllegalArgumentException is thrown	IllegalArgumentException was thrown	Confirm that error handling is robust
6	Test error handling for negative total votes	Total Seats: 5, Total Votes: -100	IllegalArgumentException is thrown	IllegalArgumentException was thrown	Confirm that error handling for invalid votes is robust

Post condition(s) for Test:
The state of the ClosedListElection object should reflect the correct allocation of seats based on the election results. No residual data should interfere with subsequent tests.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __

Test Date: 2024.04.20

Test Case ID#: OpenListElection_Test1

Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun

Test Description:
This test case evaluates the OpenListElection class's functionalities, focusing on the accurate processing of votes for individual candidates and their respective parties, and the appropriate allocation of seats based on these votes. Specific methods tested include conductElection(), calculateSeats(), and handling of edge cases such as invalid votes or empty ballots.

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

Automated: yes ✓ no

Results: Pass ✓ Fail

Preconditions for Test:
1. **Proper ballot data must be provided, and all parties and candidates must be correctly initialized within the system.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Conduct an election with valid ballots	Ballots: ["1,0,1,0", "1,0,0,1", "0,1,1,0"]	Proper tally of votes for candidates and parties	Proper tally of votes for candidates and parties	This checks vote processing and seat allocation logic
2	Verify the seat allocation		Seats correctly allocated based on votes	Seats correctly allocated based on votes	Ensures proportional representation
3	Test handling of an empty ballot	Ballots: ["", " "]	No votes counted	No votes counted	Verifies system resilience to empty input
4	Test election with invalid vote inputs	Ballots: ["1,1,-1,0", "2,0,0,1"]	Invalid votes ignored, valid votes processed correctly	Invalid votes ignored, valid votes processed correctly	Checks robustness of vote parsing
5	Test exception handling for invalid total seats	Total Seats: -1, Total Votes: 100	IllegalArgumentException thrown	IllegalArgumentException thrown	Ensures proper error handling
6	Test election process without any votes	Total Seats: 5, Total Votes: 0	No seats allocated	No seats allocated	Tests system behavior with zero votes

Post condition(s) for Test:
The state of the OpenListElection object should accurately reflect the results of the vote tally and seat allocation process. The system should remain stable and responsive even when faced with edge cases.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit _✓_ System __ **Test Date: 2024.04.21**

Test Case ID#: MPOMV_Test1 **Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun**

Test Description:
This test case examines the MPOMV class's ability to handle Multi-Party Open-Mixed Voting, focusing on individual candidate vote tallying, seat allocation based on those votes, and the handling of edge cases like no votes or excess ballots. Methods tested include conductElection(), calculateSeats(), and error handling for invalid inputs such as invalid total seats.

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

Automated: yes ✓ no

Results: Pass ✓ Fail

Preconditions for Test:

1. **Candidates and parties must be correctly initialized and linked. Ballot data provided must match expected formats, and the total number of seats must be properly set.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Test initialization of MPOMV with valid inputs	Total Seats: 2, Candidates: 3, Votes: 100	MPOMV instance correctly initialized with specified values	MPOMV instance correctly initialized with specified values	Validates basic instantiation and parameter parsing
2	Test election with no votes	Ballots: []	No votes should result in zero votes for each candidate	No votes result in zero votes for each candidate	Edge case for handling empty input
3	Test election processing with valid votes	Ballots: ["1,0,0", "0,1,0", "0,0,1", "1,0,0"]	Correct vote counts per candidate	Correctly counting for per candidate	Normal case for vote processing
4	Test calculation of seats based on votes		Seats allocated based on vote totals	Seats allocated based on vote totals	Ensure proportional seat allocation
5	Test handling of invalid total seats (negative value)	Total Seats: -1	IllegalArgumentException thrown	IllegalArgumentException thrown	Test error handling for invalid configuration
6	Conduct election with edge case of missing votes	Ballots: ["1"]	Missing votes for some candidates handled gracefully	Missing votes for some candidates handled gracefully	Handling incomplete data scenarios
7	Conduct election and ignore extra votes beyond candidates	Ballots: ["1,0,0,1"]	Extra votes beyond candidate count ignored	Extra votes beyond candidate count ignored	Ensures votes do not exceed candidate count

Post condition(s) for Test:

The state of the MPOMV object should reflect accurate and proportional representation of votes to seats. The system should gracefully handle both normal and exceptional cases without crashing.

Project Name: Project 2: Voting System **Team# 22**

Test Stage: Unit __ System _✓_

Test Date: 2024.04.23

Test Case ID#: MPOMV_Test1

Name(s) of Testers: Ruirui Xu, Naiqi Jiang, Jiahao Sun

Test Description:

This test suite evaluates the end-to-end functionality of the ElectionManager class across different election types such as Closed Party List (CPL), Open Party List (OPL), Multi-Party Open-Mixed Voting (MPO), and Majority Vote (MV).
The tests cover the complete process from loading ballot data, conducting elections, finalizing results, to generating audit reports, ensuring the system handles real-world scenarios effectively.

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

Automated: yes ✓ no

Results: Pass __✓___ Fail_____

Preconditions for Test:

1. **Properly formatted ballot data files must be accessible; the ElectionManager must be correctly initialized without any pre-loaded data.**

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Test CPL election process with multiple data files	Files: CPL_Voting_1.csv, CPL_Voting_2.csv	CPL election correctly processes and audit report generated	CPL election successfully processed and audit report generated	Checks complete CPL election handling
2	Verify OPL election processing	File: OPL_Voting_1.csv	OPL election data is correctly processed and results are detailed in audit report	OPL election data was correctly processed and results were detailed in audit report	Ensures OPL-specific functionalities are tested
3	Conduct MPO election from start to finish	File: MPO_Voting_1.csv	MPO election is conducted and audited with accurate seat and vote counts	MPO election was conducted and audited with accurate seat and vote counts	Verifies handling of MPO election complexities
4	Execute MV election and	File: MV_Voting_1.csv	MV election is processed and	MV election was processed and results were	Tests MV election flow and

	validate results		results are reflected in audit file	reflected in audit file	outcome verification
5	Confirm audit files contain correct election details	Relevant audit files for each test	Audit files include specific strings confirming election details	Audit files did include specific strings confirming election details	Critical for validating output correctness

Post condition(s) for Test:

The ElectionManager object should reflect accurately processed election data for each type tested. Audit files are generated as expected, and system integrity is maintained throughout testing.