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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: VoteCounter_Test	**Tester Name(s):** Michael Huang
Description: Testing the addVoteCount and setVoteCount methods of the VoteCounter abstract class. This includes verifying the ability to accurately add to and set specific vote counts for both entities.	**File:** VoteCounterTest.java **Methods/Functions:** addVoteCount() to test cumulative addition to vote counts. setVoteCount() to test setting vote counts to specific values.
Automated: yes	**Results**: pass
Preconditions: The addVoteCount and setVoteCount methods are also correctly implemented in the VoteCounter class.	**Postconditions:** The state of the VoteCounter objects reflects the accurate vote count after using both the addVoteCount and setVoteCount methods, demonstrating that these shared behaviors function correctly.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	addVoteCount	Initial vote count = 0, then add 11, 5, 99	Sequential vote counts of 11, 16, and 115	11,16,115	Test the ability to add vote counts on top of the current count value.
2	setVoteCount	Set vote counts to 11, then 5, then 99	Vote counts are correctly set to 11, 5, and 99	11,5,99	Test the ability to test the setting vote counts

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: Candidate_Test	**Tester Name(s):** Michael Huang
Description: Testing the Candidate class (data storage) for its ability to store and access data.	**File:** CandidateTest.java **Methods/Functions:** cons() for testing the constructor. giveSeat() for testing the giveSeat method. testToString() for testing the toString method.
Automated: yes	**Results**: pass
Preconditions: The Candidate class and its methods (constructor, giveSeat, isSat, toString) are correctly implemented.	**Postconditions:** The Candidate object states are correctly modified and validated after each test method execution.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	cons(): Instantiate a Candidate with the name "candidateName" and verify the name, seat status, and vote count.	"candidateName"	candidate.getName() should return "candidateName". candidate.isSat() should return false. candidate.getVoteCount() should return 0.	As expected.	Testing the Candidate class constructor initialization.
2	giveSeat(): Call giveSeat on a Candidate object and verify the seat status changes appropriately.	"candidateName"	Before giveSeat(), candidate.isSat() should return false. After the first giveSeat(), candidate.isSat() should return true. After the second giveSeat(), candidate.isSat() should still return true.	As expected.	Ensures giveSeat method correctly updates the candidate's seat status and is idempotent.
3	testToString(): Verify the toString method returns the correct string representation of a Candidate.	"candidateName" true 99	"Candidate{hasSeat=true, name='candidateName', voteCount=99}"	As expected.	Testing toString method after changing the seat status and vote count.

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: Party_Test	**Tester Name(s):** Michael Huang
Description: Testing the Party class (data storage) for its ability to store and access data.	**File:** PartyTest.java **Methods/Functions:** cons1() for testing the constructor with name. cons2() for testing the constructor with name & independence. setIndep() for testing the setIndep method. testToString() for testing the toString method.
Automated: yes	**Results**: pass
Preconditions: The Party class and its methods (constructor, setIndep, isIndep, toString) are correctly implemented.	**Postconditions:** The Party object states are correctly modified and validated after each test method execution. Object states are correctly modified and validated after each test method execution, demonstrating the proper functionality of constructors, status setters, and representation methods.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	cons1(): Instantiate a Party with the name "partyName" and verify the name, independence, and vote count	"partyName"	party.getName() should return "partyName". party.isIndep() should return false. party.getVoteCount() should return 0.	As expected.	Testing the Party constructor initialization.
2	cons2(): Instantiate a Party with independence true and the name "partyName", then verify the name, independence, and vote count	true "partyName"	party.getName() should return "partyName". party.isIndep() should return true. party.getVoteCount() should return 0.	As expected.	Testing the Party constructor initialization.
3	setIndep(): Call setIndep() on a Party and verify the isIndep flag is properly set.	"partyName"	After setIndep(true), party.isIndep() should return true. After a second setIndep(true), party.isIndep() should still return true. After setIndep(false), party.isIndep() should return false.	As expected.	Testing the setIndep() method
4	testToString(): Verify the toString method returns the correct string representation of a	"partyName" true 99	"Party{indep=true, name='partyName', voteCount=99}"	As expected.	Testing toString method after changing the independence and vote count.

	Party.				

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: FileInput_Test	**Tester Name(s):** Michael Huang
Description: Testing the functionality of the FileInput class, including handling valid file paths for different file types, reaction to empty and invalid file paths, and user input handling for file names.	**File:** FileInputTest.java **Methods/Functions:** testOPL() for testing OPL file recognition. testCPL() for testing CPL file recognition. testEmptyString() for testing empty string input. testGarbageInput() for testing invalid file name input. prompt1() for testing user prompting and input handling.
Automated: yes	**Results**: pass
Preconditions: The FileInput class implements methods for detecting OPL and CPL files, handling file names, and prompting user input.	**Postconditions:** The FileInput object correctly processes and validates file paths based on the provided test cases, demonstrating proper error handling and file type detection.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	(testOPL) Verify that we can input an OPL file, read its lines, and close it.	"Project1/testing/testfiles/OPLexample.txt"	fin.isOPL() returns true. fin.getLine() return "2". fin.close() does not error.	As expected.	Ensures OPL files are correctly handled.
2	(testCPL) Verify that we can input an CPL file, read its lines, and close it.	"Project1/testing/testfiles/CPLexample.txt"	fin.isOPL() returns false. fin.getLine() return "3". fin.close() does not error.	As expected.	Ensures CPL files are correctly handled and not confused with OPL files.
3	(testEmptyString) Verify that passing an empty string as a file path throws a RuntimeException.	""	RuntimeException is thrown.	RuntimeException is thrown.	Ensures that empty file paths are correctly handled as errors.
4	(testGarbageInput) Verify that passing an invalid file name throws a RuntimeException.	"garbageFILENAME0293v8woemlrivjm02"	RuntimeException is thrown	RuntimeException is thrown	Ensures invalid file names are correctly handled as errors
5	(prompt1) Verify that user input for file names is correctly	User input sequence "garbage1", "garbage2","Project1/testing/testfi	"Project1/testing/testfiles/OPLexample.txt" is correctly captured and	As expected.	Demonstrates user prompting and input handling functionality.

	handled.	les/OPLexample.txt"(valid path)	stored.		

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: CPLAllocateSeats_Test	**Tester Name(s):** Kevin Yang
Description: Testing functionality of AllocatePartySeats, and allocateCandidates. Varies the following information: 1. FirstAllocation < candidates vs FirstAllocation >= candidates 2. remainingSeats < partyCount vs remainingSeats >= partyCount 3. FirstAllocation < candidates from each party vs FirstAllocation >= candidates from each party 4. TieStart=SeatsLeft-1 vs TieStart<SeatsLeft-1 5. TieEnd=SeatsLeft-1 vs TieEnd>SeatsLeft-1 For allocateCandidates, varies: 1. TotalAllocation of Seats=0 vs >0	**File:** CPLAllocateSeatsTest.java **Methods/Functions:** allocatePartySeats() for its ability to input correct information into allocationData CPL.assignCandidateSeats() for its ability to give seats to the proper candidates
Automated: Yes	**Results**: Pass
Preconditions: There are more than 0 ballots ballotCount, seatCount, partyCount, candidateCount and voteCount for all parties is correct, candidates are listed in the correct order, default for isSat is false.	**Postconditions:** allocationData has all information about seat allocation as outlined in the project description. The boolean hasSeat is properly assigned to all candidates

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Testing with the variations: 1. FirstAllocation < candidates 2. remainingSeats < partyCount 3. FirstAllocation < candidates from each party 4. TieStart=SeatsLeft-1 5. TieEnd=SeatsLeft-1	CPLexample.csv	allocationData= {{3, 2, 0, 2, 1, 1}, {1, 0, 0, 0, 0, 0}, {0, 2, 0, 2, 1, 1}, {0, 1, 0, 1, 0, 0}, {1, 1, 0, 1, 0, 0}}	As expected.	Allocation Data is filled
2	Testing with the variations: 1. FirstAllocation >= candidates 2. remainingSeats >= partyCount	CPLexample2.csv	allocationData= {{0, 1, 2, 8}, {0, 0, 1, 1},	As expected.	Allocation Data is filled

	3. FirstAllocation < FirstAllocation+Second Allocation >= candidates from each party 4. TieStart=SeatsLeft-1 5. TieEnd=SeatsLeft-1 Note that different conditions are testing using different parties		{0, 1, 0, 6}, {3, 3, 2, 0}, {3, 3, 3, 1}}		
3	Testing with the variations: 1. FirstAllocation < candidates 2. remainingSeats < partyCount 3. FirstAllocation+Second Allocation < candidates from each party 4. TieStart<SeatsLeft-1 5. TieEnd>SeatsLeft-1 Note that different conditions are testing using different parties	CPLexampleTies.csv	allocationData= {{2, 6, 2, 0, 6, 2}, {0, 1, 0, 0, 1 ,0}, {2, 0, 2, 0, 0, 2}, {1, 0, 0, 0, 0, 0}, {1, 1, 0, 0, 1, 0}} or {{2, 6, 2, 0, 6, 2}, {0, 1, 0, 0, 1 ,0}, {2, 0, 2, 0, 0, 2}, {0, 0, 0, 0, 0, 1}, {0, 1, 0, 0, 1, 1}} or {{2, 6, 2, 0, 6, 2}, {0, 1, 0, 0, 1 ,0}, {2, 0, 2, 0, 0, 2}, {0, 0, 1, 0, 0, 0}, {0, 1, 1, 0, 1, 0}}	As expected.	Allocation Data is filled
4	Tests both allocation of seats for a party=0 and allocation of seats for a party>0 using different parties. Parties 1, 2 and 4 get seats while parties 3, 5 and 6 get no seats.	CPLexample.csv	candidates.get(0).get(0).isSat() is true candidates.get(0).get(1).isSat() is false candidates.get(0).get(2).isSat() is false candidates.get(1).get(0).isSat() is true candidates.get(1).get(1).isSat() is false candidates.get(1).get(2).isSat() is false candidates.get(2).get(0).isSat() is false candidates.get(3).get(0).isSat() is true candidates.get(3).get(1).isSat() is false candidates.get(4).get(0).isSat() is false candidates.get(5).get(0).isSat() is false	As expected.	Data in isSat for all candidates if filled.

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: OPLAllocateSeats_Test	**Tester Name(s):** Kevin Yang
Description: Testing functionality of allocate candidates for OPL elections. Cases to vary are: seatNo=0 vs seatNo>0 TieStart=seatNo-1 vs TieStart<seatNo-1 TieEnd=seatNo-1 vs TieEnd>seatNo-1	**File:** OPLAllocateSeatsTest.java **Methods/Functions:** CPL.assignCandidateSeats() for its ability to give seats to the proper candidates
Automated: Yes	**Results**: Pass
Preconditions: There are more than 0 ballots ballotCount, seatCount, partyCount, candidateCount and voteCount for all parties is correct, voteCount for all candidates is correct, default for isSat is false.	**Postconditions:** allocationData has all information about seat allocation as outlined in the project description. The boolean hasSeat is properly assigned to all candidates

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Testing with the following variations: 1. For party 1, TieStart=TieEnd=seatNo-1 For party 2, TieStart=SeatsNo-1<TieEnd For party 3, seatNo=0.	OPLexample.csv	candidates.get(0).get(0).isSat is True, candidates.get(0).get(1).isSat is False, candidates.get(0).get(2).isSat is False, Exactly one of candidates.get(1).get(0).isSat and candidates.get(1).get(1).isSat is True, candidates.get(2).get(0).isSat is True.	As expected.	Data in isSat for all candidates if filled.
2	Testing with the following variations: 1. For party 1, TieStart=TieEnd=seatNo-1 For party 2, TieStart<SeatsNo-1<TieEnd For party 3, seatNo=0.	OPLexampleTies.csv	candidates.get(0).get(0).isSat is True, candidates.get(0).get(1).isSat is False, candidates.get(0).get(2).isSat is False, Exactly two of candidates.get(1).get(0).isSat and candidates.get(1).get(1).isSat and candidates.get(1).get(2).isSat is True, candidates.get(2).get(0).isSat is True.	As expected.	Data in isSat for all candidates if filled.

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

Test Stage: Unit	**Test Date:** 3/27/2024
Test Case ID#: ElectionMethods_Test	**Tester Name(s):** Michael Huang
Description: Test all non-major Election class methods: generateAllocationTable(), generateWinnerList(), and all getters.	**File:** ElectionTest.java **Methods/Functions:** getters(): test the getter methods for accuracy in returned data. generateAllocationTable(): tests the Election method by the same name. generateWinnerList(): tests the Election method by the same name.
Automated: yes	**Results**: pass
Preconditions: Election class fields are not contradictory, i.e. allocationData and parties and candidates agree in both number and data.	**Postconditions:** Methods exhibit correct functionality and return the correct data/object/Strings.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	getters(): tests the getters for ballot, party, candidate, and seat counts, as well as allocationData, parties, candidates.	See test file for data.	All data is correctly formatted and equivalent. No object getters return the same object, only deep copies.	As expected.	All getters are accurate and satisfy encapsulation, as no internal objects are exposed.
2	generateElectionInfo(): tests the basic election information output as a String for display/audit	"Bob" from party "p2" has won the only seat	"Seat Winners and their Party Affiliation:\n - Bob (p2)\n"	As expected.	Winners are properly formatted.
3	generateAllocationTable(): tests the allocation data table output as a String for display/audit	Alice received 100 votes, Bob received 200 and won the only seat. allocationData = {{100, 200}, {0, 0}, {100, 200}, {0, 1}, {0, 1}}	Vote proportions are properly calculated, and data is properly formatted in tabular format.	As expected.	Allocation data is properly formatted.

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

Test Stage: Unit	**Test Date:** 3/27/2024
Test Case ID#:	**Tester Name(s):** Kevin Yang
Description: Tests generatePartyCandidateList() for both OPLElection and CPLElection	**File:** PartyCandidateListTest.java **Methods/Functions:** OPLElection.generatePartyCandidateList(): Tests generatePartyCandidateList in the OPL election class CPLElection.generatePartyCandidateList(): Tests generatePartyCandidateList in the CPL election class
Automated: yes	**Results**: pass
Preconditions: The correct data is in parties, candidates, and allocationData, and the data in these fields is not contradictory.	**Postconditions:** Returns a string that is guaranteed to hold the proper formatting

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Tests generatePartyCandidateList for an OPL election to make sure it returns the proper string for the display and the audit file	See test file for data.	generatePartyCandidateList returns: “Parties & Candidates: - Party: p1, Votes: 100 (33.33% of total) - Candidate: Alice, Votes: 100 (33.33% of total) - Party: p2, Votes: 200 (66.67% of total) - Candidate: Bob, Votes: 200 (66.67% of total) “	As expected	All printed information is properly formatted
2	Tests generatePartyCandidateList for a CPL election to make sure it returns the proper string for the display and the audit file	See test file for data.	generatePartyCandidateList returns: “Parties & Candidates: - Party: p1, Votes: 100 (33.33% of total) - Candidate: Alice - Party: p2, Votes: 200 (66.67% of total) - Candidate: Bob “	As expected	All printed information is properly formatted

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: Tiebreak_Test	**Tester Name(s):** Michael Huang
Description: Testing the tiebreak method within the Election class, specifically ensuring it correctly chooses a set number of unique, sorted indices within a specified range.	**File:** TiebreakTest.java **Methods/Functions:** testChoose(int n, int r) performs the tiebreak test testN2R1(), testN8R5(), testN8R1(), testN1R1() are test methods utilizing testChoose with specific values for n and r.
Automated: yes	**Results**: pass
Preconditions: The Election class and its tiebreak method are implemented.	**Postconditions:** The tiebreak method within the Election class successfully selects the correct number of unique and sorted indices from the specified range.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Verify the tiebreak method can correctly choose 1 index from 2.	n=2, r=1	A single unique index is chosen. Index within the range [0, n).	As expected.	Tests the method's ability to select an index or indices from the given sets.
2	Verify the tiebreak method can correctly choose 5 indices from 8.	n=8, r=5	Five unique indices are chosen and sorted. All indices within range [0, n).	As expected.	
3	Verify the tiebreak method can correctly choose 1 index from 8.	n=8, r=1	A single unique index is chosen. Index within the range [0, n).	As expected.	
4	Verify the tiebreak method can correctly choose 1 index from 1.	n=1, r=1	The only index (0) is chosen.	As expected.	

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: ProcessBallots_Test	**Tester Name(s):** Michael Huang
Description: Tests the ability of the CPLElection and OPLElection to parse the election data correctly from input files.	**File:** ProcessBallotTest.java **Methods/Functions:** CPLbasic(): uses CPL data format OPLbasic(): uses OPL data format OPLshuffle(): uses OPL data format, but candidates are not ordered in chunks by party.
Automated: yes	**Results**: pass
Preconditions: The ballot file exists, can be read, and is properly formatted.	**Postconditions:** The processBallotData() implementations in CPLElection and OPLElection properly set all relevant information fields and Party/Candidate lists and vote counts.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	CPLbasic(): Process the writeup-provided example CPL election.	CPLexample.csv	Seat, ballot, candidate, and party counts are correct. All parties have correct vote counts. Parties and candidates have the correct names in the correct order and structure	As expected.	Tests CPL election ballot format parsing
2	OPLbasic(): Process the writeup-provided example OPL election.	OPLexample.csv	Seat, ballot, candidate, and party counts are correct. All parties and candidates have correct vote counts. Parties and candidates have the correct names in the correct order and structure	As expected.	Tests OPL election ballot format parsing
3	OPLshuffle(): Process the election data when candidates aren't necessarily listed in blocks by party.	OPLexampleShuffled.csv	Seat, ballot, candidate, and party counts are correct. All parties and candidates have correct vote counts. Parties and candidates have the correct names in the correct order and structure	As expected.	Tests OPL election ballot format parsing, when candidates aren't necessarily listed in blocks by party.

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

Test Stage: Unit	**Test Date:** 3/25/2024
Test Case ID#: AuditFileGen_Test*	**Tester Name(s):** Oguzhan Goktug Poyrazoglu
Description: This test case evaluates the generateAuditFile method's ability to generate a comprehensive audit file containing election information, party/candidate lists, LRA calculations, and winner details.	**File:** **Methods/Functions:** generateAuditFile()
Automated: no	**Results**: pass
Preconditions: - The election data is loaded into the system and the system processes the election and has all the required valid information to generate the audit file.	**Postconditions:** The generated audit file remains in the specified directory for future reference. No changes are made to the election data by this process

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Invoke the generateAuditFile method()	“CPLexample.txt”	An audit file is successfully created in the specified directory.	As expected.	Tests the audit file creation in the specified directory
2	Verify that an audit file is created with a timestamped name	Created Audit File Name	electionType + "_AuditFile_" + yyyy-MM-dd_HH-mm-ss-SSS(date) + ".txt"	As expected.	Tests the file's timestamp format for uniqueness and traceability.
3	Open the audit file and review its contents for accuracy.	Created Audit File	Valid and unchanged information of election results	As expected.	Tests the content of the audit file.

* This test has been controlled manually.

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

Test Stage: System	**Test Date:** 3/25/2024
Test Case ID#: CPLElection_Test	**Tester Name(s):** Oguzhan Goktug Poyrazoglu
Description: System testing of the CPLElection class, focusing on processing ballot data from CPL format files, allocating party seats, and assigning candidate seats. Validates the functionality with files of varying complexity and size.	**File:** CPLElectionSystemTest.java **Methods/Functions:** test1() for testing with "CPLexample.txt" test2() for testing with "CPLexample2.txt" testLarge() for testing with "CPLexamplelarge.txt"
Automated: yes	**Results**: pass
Preconditions: The CPLElection, FileInput, Party, and Candidate classes are correctly implemented. Test files "CPLexample.txt", "CPLexample2.txt", and "CPLexamplelarge.txt" are available and correctly formatted.	**Postconditions:** The CPLElection system demonstrates the capability to accurately process ballot data from CPL format files, correctly allocate seats to parties based on vote counts, and assign those seats to candidates within the parties. The system handles files of varying complexity and size.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Processing CPL ballot data and assigning seats for a simple scenario.	"CPLexample.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected.	Ensures every piece of the election module works as expected.
2	Testing with a second set of data to validate allocation logic and candidate seating.	"CPLexample2.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected.	Ensures the seat allocation logic with a different dataset
3	Large data set processing and allocation testing to ensure system scalability and accuracy.	"CPLexampleLarge.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected.	Ensures the system's capacity to deal with the large dataset

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

Test Stage: System	**Test Date:** 3/25/2024
Test Case ID#: OPLElection_Test	**Tester Name(s):** Abbas Mohamud
Description: System testing of the OPLElection class, focusing on processing ballot data from OPL format files, allocating party seats, and assigning candidate seats. Validates the functionality with files of varying complexity and size.	**File:** OPLElectionSystemTest.java **Methods/Functions:** test1() for "OPLexample.txt". test2() for "OPLexample2.txt". testLarge() for "OPLexamplelarge.txt".
Automated: yes	**Results**: pass
Preconditions: The OPLElection, FileInput, Party, and Candidate classes are correctly implemented. Test files "OPLexample.txt", "OPLexample2.txt", and "OPLexamplelarge.txt" are available and properly formatted.	**Postconditions:** The OPLElection system demonstrates the capability to accurately process ballot data from OPL format files, correctly allocate seats to parties based on vote counts, and assign those seats to candidates within the parties. The system handles files of varying complexity and size.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Processing OPL ballot data and assigning seats for a straightforward scenario.	"OPLexample.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected	Ensures every piece of the election module works as expected.
2	Processing a second, more complex set of OPL data to validate allocation and candidate seating logic.	"OPLexample2.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected	Ensures the seat allocation logic with a different dataset
3	Large-scale OPL data processing and seat allocation testing to ensure system scalability and accuracy.	"OPLexampleLarge.txt"	Correct counts for seats, ballots, parties, and candidates; accurate vote counts and seat assignments.	As expected	Ensures the system's capacity to deal with the large dataset

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

Test Stage: System	**Test Date:** 3/25/2024
Test Case ID#: Time_Test	**Tester Name(s):** Michael Huang
Description: Evaluating the performance of the AESDriver when processing large-scale Closed Party List (CPL) and Open Party List (OPL) election data files. The key metric for assessment is the time taken to complete processing, which is expected to be under 4 minutes for each test case.	**File:** TimeTest.java **Methods/Functions:** testCPL() for timing the processing of CPLElection testOPL() for timing the processing of OPLElection
Automated: yes	**Results**: pass
Preconditions: The AESDriver class is correctly implemented and capable of processing both CPL and OPL formatted files. The test files "CPLexamplelarge.txt" and "OPLexamplelarge.txt" are available and properly formatted for large-scale data processing.	**Postconditions:** The AESDriver demonstrates high efficiency in processing large-scale CPL and OPL files, completing tasks within the expected time frame of 100,000 ballots in under 4 minutes. This performance testing validates the system's capability to handle extensive data loads.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Timing the AESDriver's processing of a large CPL file.	"CPLexampleLarge.txt"	Processing time is less than 4 minutes.	As expected	Ensures the system can process both CPL and OPL elections within the expected time.
2	Timing the AESDriver's processing of a large OPL file.	"OPLexampleLarge.txt"	Processing time is less than 4 minutes.	As expected	