| **Project Name: Project 1: Voting System Team# 9** | |
| --- | --- |
| **Test Stage: Unit** | **Test Date: 03/26/2021** |
| **Test Case ID#: IRObjTest** | **Name(s) of Testers: Isaac** |
| **Test Description:**  We are testing the functionality of the IR class. Each method will be unit tested and using asserts from JUnit. We are testing that we are getting the correct value from each method within the IR class to test functionality within the system, because IR contains all the methods to run an election. |  |
| **Automated: Yes** | **Indicate where are you storing the tests (what file) and the name of the method/functions being used.**  The tests are stored in IRTest.java. The methods tested are: parse(), tieBreaker(), runIR(), initLog(), eliminate(), reallocate() |
| **Results: Pass** |  |
|  |  |
| **Preconditions for Test:** Must have test files within the same directory of IRTest.java, because IRTest will use test election csv files to test the output of each method and the election. | |

| **Step**  **#** | **Test Step**  **Description** | **Test**  **Data** | **Expected**  **Result** | **Actual**  **Result** | **Notes** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 1 | testParse1()  This will test if the parse() function correctly parses the number of candidates, ballots, and parties. The IR constructor calls parse() inside of it. | static File testFile1 = new File("src/votecountsystem/irTest.csv");  IR ir = new IR(testFile1, false); | assertEquals(3, ir.getCandidates());  assertEquals(40, ir.getBallots());  assertEquals(3, ir.getParties().length); | assertEquals(3, ir.getCandidates()); = PASS 3  assertEquals(40, ir.getBallots()); = PASS 40  assertEquals(3, ir.getParties().length); = PASS 3 | All tests passed |
| 2 | testParseWithMaxCand()  This tests the parse with the max number of candidates (10), and 50 ballots, and number of parties in the parties array. | static File testFile2 = new File("src/votecountsystem/irTestFile1.csv");  IR ir = new IR(testFile2, false); | // testFile2 -> irTestFile1.csv  // contains 10 candidates (max), 50 votes, 10 parties  IR ir = new IR(testFile2, false);  assertEquals(10, ir.getCandidates());  assertEquals(50, ir.getBallots());  assertEquals(10, ir.getParties().length); | assertEquals(10, ir.getCandidates()); = PASS 10  assertEquals(50, ir.getBallots()); = PASS 50  assertEquals(10, ir.getParties().length); = PASS 10 | All tests passed |
| 3 | testRunIR()  This will test the runIR() method. Billy is the winner of testFile3, so runIR() should return a string saying Billy won. | static File testFile3 = new File("src/votecountsystem/irtest.csv");  IR ir = new IR(testFile3, false); | String result = "Billy won with 0 votes";  assertEquals(result, ir.runIR()); | String result = "Billy won with 0 votes";  assertEquals(result, ir.runIR()); = PASS “Billy won with 0 votes” | All tests passed |
| 4 | testTieBreaker()  Tests the tieBreaker() method. Should return the number in the arraylist that will be removed. | static File testFile3 = new File("src/votecountsystem/irtest.csv");  IR ir = new IR(testFile3, false);  ArrayList<Integer> arrList = new ArrayList<Integer>(); | arrList.add(1);  assertEquals(ir.tieBreaker(arrList), 1); | arrList.add(1);  assertEquals(ir.tieBreaker(arrList), 1); = PASS 1 | All test passed |
| 5 | testInitLogIntegerVariables()  This will test the initialization of the EventLogger in the IR class. We are creating a new EventLogger object to compare with the EventLogger that is created in the IR instance. This will only test for simple member variables that aren't arrays (int and strings) | static File testFile3 = new File("src/votecountsystem/irtest.csv");  IR ir = new IR(testFile3, false);  String[] names = {"Billy", "Sandra"};  char[] parties = {'D','I'};  IR ir = new IR(testFile3, false);  EventLogger expectedLog = new EventLogger("IR", 2, names, parties, 0, 0);  EventLogger actualLog = ir.initLog(); | assertEquals(expectedLog.getType(), actualLog.getType());  assertEquals(expectedLog.getCandidates(), actualLog.getCandidates());  assertEquals(expectedLog.getNumOfBallots(), actualLog.getNumOfBallots());    assertEquals(expectedLog.getType(), actualLog.getType());  assertEquals(expectedLog.getType(), actualLog.getType()); | assertEquals(expectedLog.getType(), actualLog.getType()); = PASS “IR”  assertEquals(expectedLog.getCandidates(), actualLog.getCandidates()); = PASS 2  assertEquals(expectedLog.getNumOfBallots(), actualLog.getNumOfBallots()); = PASS 0    assertEquals(expectedLog.getType(), actualLog.getType()); = PASS “IR”  assertEquals(expectedLog.getType(), actualLog.getType()); = PASS “IR” | All tests passed |
| 6 | testInitLogArrayVariables()  This will test the initialization of the EventLogger in the IR class. We are creating a new EventLogger object to compare with the EventLogger that is jreated in the IR instance. This will test for each element in the member variables that are arrays. It will go through each element in the candidate{names array and see if it matches the expected value for the first for loop.  Then go through each element in the parties array and see if it matches the expected values | static File testFile3 = new File("src/votecountsystem/irtest.csv");  IR ir = new IR(testFile3, false);  String[] names = {"Billy", "Sandra"};  char[] parties = {'D','I'};  IR ir = new IR(testFile3, false);  EventLogger expectedLog = new EventLogger("IR", 2, names, parties, 0, 0);  EventLogger actualLog = ir.initLog(); | for (int i = 0; i < actualLog.getEventCandNames().length; i++)  { assertEquals(expectedLog.getEventCandNames()[i], actualLog.getEventCandNames()[i]);  }  for (int i = 0; i < actualLog.getParties().length; i++)  { assertEquals(expectedLog.getParties()[i], actualLog.getParties()[i]);  } | for (int i = 0; i < actualLog.getEventCandNames().length; i++) = PASS  { assertEquals(expectedLog.getEventCandNames()[i], actualLog.getEventCandNames()[i]);  }  for (int i = 0; i < actualLog.getParties().length; i++)  { assertEquals(expectedLog.getParties()[i], actualLog.getParties()[i]); = PASS  } | All tests pass |
| 7 | testEliminate()  This will test the eliminate() method in the IR class. Candidates at initialization should not be eliminated, then as eliminate() is called on a candidate, the candidate should be eliminated() (the candidate's eliminate boolean turns true). This test will test the first two candidates.  At first, Candidates are not eliminated at the beginning, therefore, candidate.isEliminated() should be false.  Then call the eliminate() method on the candidates to eliminate the candidate.  Then candidate.isEliminated() should return true. | static File testFile1 = new File("src/votecountsystem/irTest.csv");  IR ir = new IR(testFile1, false); | for (int i = 0; i < ir.getCandNames().length; i++)  { assertEquals(false, ir.getCandNames()[i].isEliminated());  }  for (int i = 0; i < ir.getCandNames().length; i++)  { ir.eliminate(ir.getCandNames()[i]);  }  for (int i = 0; i < ir.getCandNames().length; i++)  {  assertEquals(true, ir.getCandNames()[i].isEliminated());  } | for (int i = 0; i < ir.getCandNames().length; i++)  {  assertEquals(false, ir.getCandNames()[i].isEliminated()); = PASS false  }  for (int i = 0; i < ir.getCandNames().length; i++)  { ir.eliminate(ir.getCandNames()[i]);  }  for (int i = 0; i < ir.getCandNames().length; i++)  {  assertEquals(true, ir.getCandNames()[i].isEliminated()); = PASS true  } | All tests passed |
| 8 | testParse2()  This test will test a csv file with candidates, but zero votes. This should count 0 ballots. | static File testFile3 = new File(“src/votecountsystem/irtest.csv”);  IR ir = new IR(testFile3, false); | assertEquals(2, ir.getCandidates());  assertEquals(0, ir.getBallots());  assertEquals(2, ir.getParties().length); | assertEquals(2, ir.getCandidates()); = PASS 2  assertEquals(0, ir.getBallots()); = PASS 0  assertEquals(2, ir.getParties().length); = PASS 2 | All tests passed |
| 9 | testReallocate()  This will test the reallocate() method in the IR class. To test, we will keep track of the Ballot ID that was assigned to Billy (the first candidate), then we will eliminate Billy, the ballot will be reallocated to the next candidate.  We know from the CSV file that this tracked ballot will be reallocated to Sandra. We will go through each of Sandra's  ballots to see if the tracked Ballot ID was reallocated correctly. | static File testFile1 = new File("src/votecountsystem/irTest.csv");  boolean check = false;  IR ir = new IR(testFile1, false);  int expectedBallotID = ir.getCandNames()[0].getVotes().get(0).getID(); | // Eliminate Billy  ir.eliminate(ir.getCandNames()[0]);  // reallocate the votes that belonged to Billy. We know from the CSV file that this tracked ballot will be reallocated to Sandra. We will go through each of Sandra's ballots to see if the tracked Ballot ID was reallocated correctly.  ir.reallocate(ir.getCandNames()[0].getVotes());  for (int i = 0; i < ir.getCandNames()[1].getVotes().size(); i++) {  if (ir.getCandNames()[1].getVotes().get(i).getID() == expectedBallotID) {  check = true;  }  }  assertEquals(true, check); | Sandra. We will go through each of Sandra's ballots to see if the tracked Ballot ID was reallocated correctly.  ir.reallocate(ir.getCandNames()[0].getVotes());  for (int i = 0; i < ir.getCandNames()[1].getVotes().size(); i++) {  if (ir.getCandNames()[1].getVotes().get(i).getID() == expectedBallotID) {  check = true;  }  }  assertEquals(true, check); = PASS true |  |

**Post condition(s) for Test:**

The IR class’ method variables will have been tested and a IR object will have the proper values when read in from the CSV file and will be able to run an election of type IR. It will return the winner as a string to the console and also create audit and results csv files.

