P01 Election Manager

Overview

The United States is heading into its presidential election season, so in the spirit of the season we'd like to begin the semester by building a simple candidate management system for an election (any election).

For this programming assignment, you'll be creating a collection of utility methods to help manage a collection of candidates running against each other. You'll be able to add a candidate (in alphabetical order), drop a candidate from the race, determine whether someone's already running, and find the winner and lowest-polling candidates. Each candidate is represented in our program as follows:

candidate **NAME**	**PARTY** affiliation	number of **VOTES**

For the purposes of your instructional staff's sanity, all examples within this assignment will be for the highly contentious race for Bestest Pokemon.

This assignment is intended to be completed using primarily your prior knowledge from **BEFORE CS300!** Plan to get started as soon as possible; more help will be available to you the earlier you seek it out.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM CT on **09/08**.
+5%	**Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **WED 09/11** *and* who pass ALL immediate tests will receive an additional 2.5 points toward this assignment, **up to a maximum total of 50 points**.
20 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
15 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability.
50 points	**MAXIMUM TOTAL SCORE**

Learning Objectives

After completing this assignment, you should be able to:

- Comfortably **implement** a procedural program using static methods in Java
- **Explain** the protocols used with oversize arrays
- Verify the correctness of a static method which uses arrays by **designing** and **implementing** a boolean tester method in Java

Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **NOT ALLOWED** for this assignment. You must complete and submit P01 individually.
- The ONLY external libraries you may use in your program are:
  `java.util.Arrays` (ONLY in the tester class)
  `java.lang.reflect.InvocationTargetException` (ONLY as provided)
- Use of *any* other packages (outside of java.lang) is NOT permitted.
- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables or constants beyond those specified in the write-up.
- You are allowed to define additional **private** helper methods.
- Only `ElectionManagerTester` may contain a main method.
- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- All other sources must be cited explicitly in your program comments, in accordance with the Appropriate Academic Conduct guidelines.
- Any use of ChatGPT or other large language models ***must be cited*** AND your submission MUST include **screenshots** of your interactions with the tool ***clearly showing all prompts and responses in full***. Failure to cite or include your logs is considered academic misconduct and will be handled accordingly.
- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

Need More Help?

Check out the resources available to CS 300 students here:
https://canvas.wisc.edu/courses/427315/pages/resources

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Appropriate Academic Conduct, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - How do I cite my sources?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P01 Election Manager**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.
2. Download one (1) Java source file from the assignment page on Canvas:
   a. **ElectionManagerTester.java** (includes a main method)
3. Create one (1) Java source file within that project's src folder:
   a. **ElectionManager.java** (does NOT include a main method)

All methods in this program will be **static** methods, as this program focuses on procedural programming.

Implementation Requirements Overview

Your **ElectionManager.java** program must contain the following methods, described in detail on this javadoc page.

- `public static boolean containsCandidate(String[][] candidates, int numCandidates, String name, String party)`
- `public static int addCandidate(String[][] candidates, int numCandidates, String name, String party, int numVotes)`
- `public static int dropCandidate(String[][] candidates, int numCandidates, String name, String party)`
- `public static String findWinner(String[][] candidates, int numCandidates)`
- `public static String findLowestPollingCandidate(String[][] candidates, int numCandidates)`

All of these methods use an ***oversize array***, defined by a two-dimensional (n x 3) array of Strings and an integer value denoting the compact number of initialized (non-null) values in the first dimension of the array. That is, if the integer value `numCandidates` is 5, I would expect there to be length-3 arrays of Strings at `candidates[0]` through `candidates[4]`, and any other indexes in `candidates` will be `null`.

These methods are UTILITY methods – the ElectionManager program does not "run", as it has no main method. Another class will create and maintain the list of candidates, and call the methods in ElectionManager when it needs to. For this assignment, you'll only be calling the methods to TEST them from your tester class.

An Aside: Using Javadocs to build a program

The Javadoc page linked above is automatically generated by the IDE, using the contents of Javadoc-style comments in the program source code. When creating this assignment, I wrote ElectionManger with comments for the class (a "class header" comment) and each method (a "method header" comment) in the following style:

```
/**
 * Determines whether the given candidate, specified uniquely by name and party,
 * is present in the given list of candidates.
 *
 * @param candidates A two-dimensional oversize array containing the current
 *     list of candidates as [name, party, numVotes]. This input value is assumed to
 *     conform to the standards of oversize arrays, and is assumed to be in alphabetical
 *     order by candidate name.
 * @param numCandidates The current size of the candidates oversize array at the
 *     time of input. This value is assumed to be accurate.
```

```
 * @param name The name of the candidate to search for
 * @param party       The party affiliation of the candidate to search for
 * @return true if a candidate with the given name AND party affiliation is present
 *     in the list; false otherwise
 */
```

Both class and method header comments begin with a `/**` (two asterisks) and end with a `*/` – note that if you only use one asterisk at the beginning, this creates a multiline comment, not a Javadoc comment.

This is important documentation, and ***you will be required to do the same*** for all classes and methods you write in this course. You are welcome to use any text from the writeups, verbatim, in your comments. You are not required to generate HTML javadoc pages, but you can if you want to!

At the top of the javadoc page you will find a summary of the data fields (when there are any) and methods in the class, and if you either click on their names or scroll down, you will find the full detailed description of what these methods are supposed to do, what parameters they expect, and what they should return.

Most – if not all – of our programming assignments this semester will use these generated Javadoc pages to communicate the requirements of the assignment to you, so if you have any questions about how to use them, this is the time to ask.

Implementation Details and Suggestions

Add the methods listed on the javadoc page as ***stub methods*** – methods containing only a default return statement – to your class file, and then turn to the tester file before you go any further.

```
public static int sampleMethod() {
  return 0; // this is an example stub method that returns an int
}
```

The only way to begin, is by beginning

A link to the **ElectionManagerTester.java** starter file can be found on the assignment page, so we haven't provided a Javadoc file for it. A few test methods have been provided for you in their entirety, along with some implementation-level comments to give you a guide in constructing your test methods.

While there are a LOT of tester methods, they should all be pretty short. Use the provided methods as a model for setting up your own, and don't be afraid to use Arrays class methods as a shortcut for verifying the state of the 2-dimensional array after a method call.

Method 1: Contains Candidate

We've provided one tester here (`testDoesContain()`) which verifies the behavior of the `containsCandidate()` method from ElectionManager when the candidate name/party we're asking about is present in the list.

Read the documentation for `containsCandidate()` and review the provided implementation for `testDoesContain()`, and then answer the following questions:

1. Which of the **test variables** from `testDoesContain()` can you use to test that a candidate is NOT present in the candidateList? Which ones will you need to change?
2. What is the expected **return value** of `containsCandidate()` for a name/party combination that is not present in the list?
    a. What if the NAME matches but not the PARTY?
3. Should `containsCandidate()` modify the array if the name/party combination is not present?

With these answers, go ahead and implement the `testDoesNotContain()` test method.

Now, implement the `containsCandidate()` method itself, and finish out `testContainsEmpty()` (which should test whether an empty candidateList – with length > 0 but size == 0 – contains any given candidate; **hint**: it does not).

If you've done everything correctly up to this point, you should be able to run ElectionManagerTester in your IDE and get output that begins with:

```
testContainsEmpty            : true
testDoesNotContain           : true
testDoesContain              : true
```

If so: hooray! Time to continue. Otherwise, check your tests and method implementation; something's going wrong and it's a good idea to fix that before you go on.

➡️You may ALSO wish to make a partial submission to Gradescope at this point, and at the end of each of the following sections – this way, you can see whether you're passing any immediate tests related to the `containsCandidate()` method, AND you'll have a secure record of your work to this point. If your computer crashes or you accidentally delete your files, you can always re-download your most recent Gradescope submission!

Method 2: Add Candidate

Again you have a provided test here (`testAddToNonEmpty()`); review it and the documentation for `addCandidate()` before you continue.

You will need to implement:

1. `testAddToEmpty` – add to a length > 0[1] but size == 0 list of candidates
2. `testAddCandidateErrors` – add a duplicate candidate or a candidate with negative votes

[1] **Note**: at no point in this assignment does a length 0 array make sense. Arrays are assumed to have length > 0.

3. `testAddToFull` – add to a list of candidates with length == size, which contains ***no*** null values

As well as the `addCandidate()` method itself. We recommend implementing ***at least one*** of the TODO testers before you implement the method itself. You're absolutely welcome to use the `containsCandidate()` method in your implementation of `addCandidate()`, by the way.

Again, verify that you pass ALL of the add method tests before you continue.

Method 3: Drop Candidate

Same deal; this time the tests you're responsible for are:

1. `testDropOnlyCandidate` – remove a candidate from a list with size == 1
2. `testDropFirstCandidate` – remove the first candidate in a list with size > 1. Verify that the resulting array has been modified appropriately – it should still be a compact, oversize array!

Method 4: Find Winner

This method returns a String value. Pay close attention to how we've analyzed the actual return value in the provided `testClearWinner()` method – MOST of YOUR tests shouldn't need to go this in-depth with the analysis, but you may wish to reference this on future assignments.

1. `testUncontestedWinner` – check the winner on a list with size == 1, which should return the single candidate's name and party with 100.0% of the votes
2. `testContingentElection` – check the winner when no one candidate has > 50% of the votes, which should return "CONTINGENT"

Method 5: Find Lowest Polling Candidate

The only twist here is that we haven't provided ANY of the tester methods, so you should DEFINITELY write at least one before you implement the method itself.

1. `testUncontestedLowestPolling` – if the list has 0 or 1 candidates, this method isn't particularly meaningful; verify that it returns "UNCONTESTED"
2. `testLowestUniqueVoteCount` – find the lowest vote count on a list with size >= 2 and all candidates have unique vote counts
3. `testLowestVoteCountTied` – find the lowest vote count on a list with size >= 2 and at least two candidates are tied for lowest number of votes

At this point, you should finally see the line at the end of your output read

```
ALL TESTS: true
```

If it does, you are done!!

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, make a final submission of your source code to Gradescope.

For full credit, please submit the following files (**source code**, *not* .class files):

- `ElectionManager.java`
- `ElectionManagerTester.java`

Additionally, if you used generative AI at any point during your development, ***you must include screenshots*** showing your FULL interaction with the tool(s).

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Students whose final submission (which must pass ALL immediate tests) is made before 5pm on the Wednesday before the due date will receive an additional 5% bonus toward this assignment. Submissions made after this time are NOT eligible for this bonus, but you may continue to make submissions until Thursday **10:00 PM Central Time** on the due date with no penalty.

Copyright notice