P04 Exceptional Elections

Overview

The election manager is back, and we're going to add a new flavor to it!

In P01, you maintained the election manager list by using an oversize array as an argument to some utility methods, which trusted that the array was being correctly maintained according to oversize array protocols – compact, and with an accurate size variable.

Now that we've unlocked object-oriented design, we're going to ***encapsulate*** all of this election management into a class, so we can be absolutely sure that it's being maintained properly. If someone tries to do something they're not allowed to do, well, that's what exceptions are for.

There will be some familiar pieces to this program but also some new functionality, so read the writeup and the provided documentation carefully.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on 10/06.
+5%	**Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **WED 10/09** *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:

- **Implement** a simple object with basic accessor and mutator methods
- **Communicate** error states using exceptions and appropriate messages, and **explain** how to appropriately handle thrown exceptions.
- **Explain** the different requirements of using a static vs. a non-static method
- **Incorporate** exception handling into your tester methods so that they can survive unusual circumstances and receive communication about error states from other methods without crashing

Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **ALLOWED** for this assignment, BUT you must have registered your partnership before this specification was released. If you did not do so, you must complete this assignment individually.

- The ONLY external libraries you may use in your program are:
  Election.java - `java.util.NoSuchElementException`
  Ballot.java - `java.util.ArrayList, java.util.NoSuchElementException`
  ElectionManagerTester.java - `java.util.Arrays,` any relevant exceptions

- 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 the `ElectionManagerTester` class 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**.

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:

- 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 **P04 Exceptional Elections**[1].
   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 three (3) Java source files within that project's src folder:
   a. **Ballot.java** (does NOT include a main method)
   b. **Election.java** (does NOT include a main method)
   c. **Candidate.java** (does NOT include a main method)

[1] Because we now have exceptions, you see.

1. Election Manager Tester

The **ElectionManagerTester.java** provided to you includes all REQUIRED tester methods, which are not comprehensive. The only testers you are ***required*** to implement for this assignment are for the methods which may throw exceptions.

You are welcome to add additional testers, but they must be **private**.

We've provided two fully-implemented tester methods. This program features some expected behavior that involves throwing exceptions, and these two methods demonstrate how you should write testers to validate that kind of behavior:

1. `testCandidateConstructorAndGetters()` demonstrates a tester for a method (a constructor in this case) which ***should not*** **throw an exception**, but *may* throw one if it is poorly implemented. In this case, if you catch any exceptions, you should consider the test to have ***failed*** and return false.

2. `testVoteExceptions()` demonstrates a tester for a method which ***should*** **throw an exception**, but *may not* throw one or may throw the *wrong type* of exception if poorly implemented. In this case, if you do NOT catch an exception or if you catch the wrong type of exception, you should consider the test to have failed and return false, but if you DO catch the expected type of exception, that's good!!

As with P01, we recommend writing at least one relevant test before you implement the class/method it would test. We speak from experience here, as debugging 1 thing is WAY easier than debugging 10 things that all depend on each other.

2. The `Candidate` data type

Where P01 represented candidates as an array of 3 Strings, this program represents each candidate as its own custom-defined object. In addition to the `name` (still a String) and `party` (also still a String), we'll now be tracking `numVotes` directly as an int.

Two candidates are considered the "same" in this version of the program if their `name` and `party` match *exactly* AND if they have received the same number of votes. As part of this class, you will override Object's `equals()` method to recognize deep copy candidates as being equal – by default, `equals()` only recognizes shallow copies.

We have also required you to add an implementation of `toString()` to Candidate, so that you can easily and meaningfully print a Candidate object to the console. The format is:

```
    name (party): numVotes
```

More information and additional methods can be found in the javadocs.

3. The `Election` data type

As in P01 this design uses a compact oversized array to hold all of our Candidates, but instead of relying on the tester class to maintain the oversize arrays properly, you'll encapsulate the relevant information and methods in an object.

More information and a complete list of required methods can be found in the javadocs.

The compact, oversize `candidates` array must have nulls in elements not being used; for full credit you must REMOVE all data not currently being used. For example, if `numCandidates` is 3 and `candidates.length` is 7 then the array's contents will be as follows:

```
{candidate1, candidate2, candidate3, null, null, null, null}
```

The name of the seat an election is for is a CONSTANT value associated with each Election object. This means you *must* initialize that value in the constructor, or your code will not compile! Do not initialize this value when you declare it; if you do so, every election will be for the same seat. However, since this value is constant, we'll make it public to skip the necessity of writing an accessor method for it.

We haven't required a `containsCandidate()` method this time, but you may still want to write one. If you do, please make it a private method – there are other ways to tell whether a Candidate is present in the election! You may need to get creative ;)

Note that you will also be overriding `toString()` and `equals()` here, but in a different way than with Candidate…

4. The `Ballot` data type

And now, the brand new part! Each Ballot maintains an ArrayList of Elections, and facilitates a VOTE.

There are two phases to the life of the Ballot class:

1. Adding new Elections to the ArrayList
2. Creating new Ballots

You'll be setting up your class to enforce that these operations take place in this order! If no Elections have been added, you cannot create a Ballot; once the first Ballot is successfully created, no more Elections can be added. This setup for our utility methods would be especially useful for use in a real world application, where you would want to finalize the elections FIRST and then begin issuing ballots for those elections.

You'll accomplish this using a combination of **class** (static) and **instance** (non-static) methods and fields.

Each Ballot object will then be able to cast ONE vote in each Election, which your class will facilitate.

More information and a complete list of required methods can be found in the javadocs. Good luck!!

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, submit your source code through Gradescope.

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

- **Candidate**.java
- **Election**.java
- **Ballot**.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 10:00PM Central Time on the due date with no penalty.

Copyright notice