P01 COVID Test Tracker

Overview

Local health officials have implemented a screening protocol for a viral infection that's running rampant, and would like our help managing the data. Residents may elect to be tested at any point, as many times as they wish; at this point, all we will know is their **unique identifier** and whether a test came back **positive** or **negative**.

We'll improve upon this model as the semester progresses, but for now we'll keep things simple. In this version of our COVID Test Tracker, you will need to:

- Add a new test to the dataset (positive or negative)
- Perform simple analytics on the population
- Retrieve information about an individual's test history
- Remove an individual's tests from the dataset

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **09/06**.
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.

Learning Objectives

The goals of this assignment are:

- Reviewing procedural programming skills:
  - Control structures (e.g. iteration and conditionals)
  - Creating and using static methods
  - Using arrays with methods
- Managing an unordered collection of data with duplicates
- Developing tests to assess code functionality
- Using the CS 300 submission and automated grading test suite

Additional Assignment Requirements and Notes

Keep in mind:

- There are NO import statements allowed for this assignment.
- You are NOT allowed to add any constants or variables outside of any method.
- You are allowed to define any local variables you may need to implement the methods in this specification.
- You are allowed to define additional **private static** helper methods to help implement the methods in this specification.
- In addition to the required test methods, we strongly recommend that you implement additional unit tests to verify the correctness of every public static method implemented in your COVIDTracker class.
- All test methods must be public static, take zero arguments, and return a boolean. These methods must be contained in the COVIDTrackerTester class.
- All methods, public or private, 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.

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:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - 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 COVID**.
   a. Ensure this project uses Java 11. Select "JavaSE-11" 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. Create two (2) Java source files within that project's src folder:
   a. COVIDTracker.java (does NOT include a main method)
   b. COVIDTrackerTester.java (includes a main method)

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

Implementation Requirements Overview

Your **COVIDTracker.java** program must contain the following methods. Implementation details, including required output formatting, are provided in later sections.

- `public static boolean addTest(String[] pos, String[] neg, String id, boolean isPos)`
  - Parses the provided isPos boolean
  - Adds the provided id at the lowest-index null reference in the corresponding array
    - Hint: You'll need to look through the array to find this, as pos and neg do not come with a true oversize array's corresponding size information
  - Returns true if successfully added, false if there was not room to add
- `public static boolean removeIndividual(String[] pos, String[] neg, String id)`
  - Removes all occurrences of the provided individual from both arrays
  - Compacts the arrays so there are no empty spaces in the middle of data
  - Returns true if one or more records were removed, false if the ID was not found in either array
- `public static String getPopStats(String[] pos, String[] neg)`
  - Returns a formatted String that includes:
    - the total number of positive and negative tests
    - the total number of unique individuals across both arrays
    - the proportion of positive tests
    - the proportion of individuals who have tested positive
- `public static String getIndividualStats(String[] pos, String[] neg, String id)`
  - Returns a formatted string that includes:
    - the total number of tests for the individual
    - the total number of positive and negative tests for the individual

Your **COVIDTrackerTester.java** program must contain *at least one* testing method for each of the methods in COVIDTracker.java. These testing methods should have no parameters and return a boolean value.

COVIDTrackerTester.java should also contain a main method, which calls each of your test methods.

COVIDTracker.java must NOT contain a main method.

Implementation Details and Suggestions

For this iteration of your COVID tracker, we will assume that the data is stored in arrays created by and stored in an external main class. Your implementation will use these arrays as provided.

Adding a Test Record

Begin by adding the following method signature (with its Javadoc-style comments) to your COVIDTracker class:

```
/**
 * Adds ID to the appropriate test array if there is room.
 * @param pos The current array of positive tests
 * @param neg The current array of negative tests
 * @param id The tested individual's unique identifier String
 * @param isPos true if the test was positive, false otherwise
 * @return true if the new record was added, false otherwise
 */
public static boolean addTest(String[] pos, String[] neg, String id,
boolean isPos) {
  return false; // stub implementation, allows code to compile
}
```

We temporarily include the return statement here to satisfy the compiler, which expects this method to return a boolean. You could submit this implementation to Gradescope! Please check for compiler errors before submitting; anything which does not compile will not pass any of the automated tests.

Before writing the addTest() implementation, create a test method in your COVIDTrackerTester class:

```
/**
 * Checks whether addTest() works as expected
 * @return true if method functionality is verified, false otherwise
 */
public static boolean testAddTest() { return false; }
```

A common trap when writing tests is to make the test as complex (if not moreso) than the code it tests, which leads to Even More Bugs and development time. Keep your tests simple!

You'll also want to cover as many *cases* as you can think of:

- Inputs: empty arrays, partially-filled arrays, completely-filled arrays, one full one empty, etc.
- Outputs: situations that lead to true, situations that lead to false

Before you begin implementing, write out the combinations of inputs and outputs that you would expect given the description of this method. You can do this in comments in the test method or with a pencil/pen on paper, but you shouldn't be writing code yet!

STOP. Write out your test cases.

Once you've got an idea of the cases you want to cover with your tests, begin constructing the inputs you will need and writing calls to the addTest() method. Check the results against your expected outputs.

For this method only, we'll provide you with our suggested implementation, which you can check against yours to get an idea of what this might look like:

```
public static boolean testAddTest() {
  // two empty arrays -> true; also checking that arrays were updated properly
  String[] pos = new String[2];
  String[] neg = new String[2];
  if (!COVIDTracker.addTest(pos, neg, "AB1234", false) || neg[0] == null)
    return false;
  if (!COVIDTracker.addTest(pos, neg, "CD2345", true) || pos[0] == null)
    return false;
  // two arrays with space -> true
  if (!COVIDTracker.addTest(pos, neg, "CD2345", false) || neg[1] == null)
    return false;
  // one full array but adding to one with space -> true
  if (!COVIDTracker.addTest(pos, neg, "EF3456", true) || pos[1] == null)
    return false;
  // one array with space but adding to full one -> false
  String[] pos2 = new String[2];
  if (COVIDTracker.addTest(pos2, neg, "EF3456", false))
    return false;
  // two full arrays -> false
  if (COVIDTracker.addTest(pos, neg, "EF3456", true))
    return false;
  return true;
}
```

You may add more specific feedback when certain tests fail—it can be useful to know exactly where things went differently to what you expected.

Notice that the provided test method helps clarify the requirements and expected behavior of the addTest() method.

PRO TIP: when you encounter a problem or question about your code, design and then implement a test method to verify your understanding of what IS happening versus what SHOULD be happening. Share the test and its results with course staff, and they can help you much more efficiently.

Next, add a call to the test method from the main method of your COVIDTrackerTester class. For example:

```
/**
 * Calls the test methods implemented in this class and displays output
 * @param args input arguments if any
 */
public static void main(String[] args) {
  System.out.println("testAddTest(): " + testAddTest());
}
```

If you run this with the stub method from before, the test method should return false (i.e. not pass). Now implement the addTest() method and check whether your implementation passes your tests.

Removing an Individual

There is not much additional detail to provide about the removeIndividual() method here; any matches of the provided ID in either the positive or negative arrays should be removed, and the remaining entries compacted.

For example, if

```
pos: {"AB1234", "CD2345", null}
neg: {"EF3456", "CD2345", "AB1234"}
id: "AB1234"
```

The resulting values for pos and neg should be as follows, and the method should return true (since there was at least one match of id in the provided arrays):

```
pos: {"CD2345", null, null}
neg: {"EF3456", "CD2345", null}
```

The order of the compacted array will NOT be tested, only that it has no nulls between non-null entries.

Display Statistics

For the two statistics methods, your returned String should look like this (note that your exact numbers and the values of the statistics will change based on the composition of your arrays; to design your tests, you should do the calculations yourself). The `"\n"` character is a newline character and should be added after each line except the final one.

`getPopStats()`:

```
"Total tests: 50\n
Total individuals tested: 27\n
Percent positive tests: 24.0%\n
Percent positive individuals: 37.037037037%"
```

`getIndividualStats()`:

```
"Total tests: 3\n
Positive: 0\n
Negative: 3"
```

If getPopStats() is called with empty arrays, the values for all totals should be 0 and all percents should be 0.0%. Be careful of divide-by-zero errors!

For an individual whose identifier does not appear in either the positive or negative arrays, the total tests, positive, and negative values will all be 0. No error message should be added.

Testing Methods and Commenting

All testing methods should be called from your main() method in COVIDTrackerTester.java, and there must be *at least one* test method for each method in COVIDTracker.

Name your test methods "test" + the name of the method being tested.

Recall ALL methods and ALL classes must have Javadoc-style header comments, and both files must have a file header comment, per the CS 300 Course Style Guide. You must also add comments within the body of your code.

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 ONLY the following files (source code, *not* .class files):

- COVIDTracker.java
- COVIDTrackerTester.java

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.

Copyright Notice