CSE11-SP21-Assignments / cse11-sp21-pa5-starter

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main cse11-sp21-pa5-starter / README.md Go to file ...

shihualu Update README.md Latest commit 05c4883 16 hours ago History

2 contributors

122 lines (92 sloc) 9.72 KB Raw Blame

CSE-11 Programming Assignment 5

Due Date: Wednesday, May 5, 11:59PM Pacific Time

Learning Goals

- Practice writing methods that calculate values from arrays using loops.
- Practice making a screencast in preparation for the take-home screencast exam.

Collaboration

Different assignments in this course have different collaboration policies. On this assignment, you can collaborate with anyone in the course, including sharing code. In your submission, give credit to all students and course staff who helped you with this assignment by noting their name and how you used their ideas or work. Note that using someone's work without giving credit to them is a violation of academic integrity.

We've added a `CREDITS.txt` to note others you worked with.

Resubmission/Late Policy

- We will not accept this PA late.
- If you did not submit this PA or did poorly (<75%), you can fix your PA and resubmit it to the PA resubsmission assignment in Gradescope to earn a maximum of 75% the points.
- If you scored higher than 75% on the origianl PA, we may grade the resubmission, but will not change your original grade.
- The resubmission will be open for 2 weeks after the PA's original due date.

Array Methods

In a file called `ArrayExamples.java`, write the following methods in a class called `ArrayExamples`. For **each**, write **at least three** tests (a test is a use of `checkExpect`) where each of the three has a different length of array used in the input.

- Write a method called `joinWith` that takes an array of `String` and a `String` separator, and returns a new String that contains the strings from the array separated by that separator. For example, for an array containing `"a"`, `"b"`, and `"c"` with separator `":"`, the result would be `"a:b:c"` (note that there's no colon at the end, just in between the elements). If the input array is empty, the method should return the empty string. If the input array contains only one string, the method should return that string.

String joinWith (String[] arr, String sep) {...}

- Write a method called `allTrue` that takes an array of `boolean` and returns `true` if all the elements in the array are `true`. If the array is empty, the method produces `true`.
- Write a method called `allWithinRange` that takes an array of `double` and two other `double`s called `low` and `high`, and returns `true` if all of the numbers in the array are between `low` and `high` (inclusive). If the array is empty, this should produce `true`. You can assume that `low ≤ high`.
- Write a class called `Pair` with two `int` fields, `a` and `b`, and include a constructor. (Add `Pair` at the top level, outside the `ArrayExamples` class). Then write a method (in `ArrayExamples`, not in `Pair`) called `maxmin` that takes an array of `int` and returns a `Pair` where the `a` field is set to the smallest integer in the array and the `b` is set to the largest. Assume the array has at least one element.
- Write a method called `earliest` that takes an array of `String`s and returns the `String` that is the earliest alphabetically (Computer scientists have a fancy name for alphabetical: **lexicographic**. You will need the `compareTo` method on Strings here. Try it out on a few examples if you're not sure what it will do!). You can assume that the array has at least one element.
- Write a method called `lookup` that takes an array of `String` called `keys`, an array of `int` called `values`, and a `String` called `key` (three total parameters). It should find the index in `keys` where the argument `key` appears, and then return the `int` stored in `values` at that index (Hint: you may want to use `.equals` or `.compareTo` for string comparison instead of `==`). If the key is not found, the method should return `-1`. You can assume that `lookup` will always be given two arrays of the same length, and that there are no duplicate strings in `keys`. Example: `keys` contains `"UCSD"`, `"UCLA"`, `"UCI"` and `values` contains `36000`, `44900`, and `33467`. For `key` `"UCI"`, it should return `33467`. For key `"Stanford"`, it should return `-1`.

Using Main and Command-line Arguments

- In a file called Longest.java, write a class called Longest. It should have a main method which prints out the longest string in the command line arguments. If no arguments were given, it should print nothing. Example:

```
$ javac Longest.java
$ java Longest which argument is the longest
argument
$ java Longest one two three four
three
$ java Longest
$
```

  You can assume that there is not a tie for the longest string's length.

- In a file called `Stats.java`, write a class called `Stats`. It should have a main method which has a different effect depending on the first command line argument. In all cases, it can assume that there will be at least two command-line arguments, and all the arguments after the first are appropriate arguments to `Double.parseDouble`. If the first argument is ...
  - `"--product"`, print the product of the provided numbers
  - `"--mean"`, print the average (mean) of the provided numbers
  - `"--total"`, print the sum of the provided numbers
  - `"--max"`, print the maximum of the provided numbers
  - `"--min"`, print the minimum of the provided numbers
  - any other string, print "Bad option " where you will replace "" with the first argument
  - *(Hint: As has been mentioned above, when comparing Strings, the `==` operator can be unreliable. Instead use `.equals` or `.compareTo`, which are in the Java String documentation.)*

  Examples:

```
$ javac Stats.java
$ java Stats --product 2 3 4
24.0
$ java Stats --mean 5 9 7
7.0
$ java Stats --total 1 9 4
14.0
$ java Stats --max 9 1 4 0
```

```
9.0
$ java Stats --min 9 1 4 0
0.0
$ java Stats --mix 3 4 5
Bad option --mix
```

Practice Screencast

For the take-home exams, you'll be recording a screencast that demonstrates your programs. The last part of this PA has you practice with this.

You will record a short video (no more than 2 minutes). Include:

- Your face and your student ID for a few seconds at the beginning. You don't have to be on camera the whole time, though it's fine if you are. Just a brief confirmation that it's you creating the video and doing the work attached to the video.
- A capture of your screen running your `Longest` program. Make sure the code of `Longest.java` AND your terminal output are shown in the video
- Show running the two examples of Longest above, with their results clearly visible in the terminal

There is a short tutorial demonstrating how to make a screencast with Zoom:

Screencast Tutorial (Credits: Joe Politz)

The associated video that was created is here:

Example Result (Credits: Joe Politz)

You don't have to make your screencast with Zoom, but Zoom is UCSD-licensed software that you have access to, so we offer the tutorials with that in mind.

Please do reach out if you run into issues with this, and try something out early so you know if it will work or not for you – you'll need it for the exam!

Submission

You will upload your video file to this link:

Upload Form

All of your Java files you will upload to Gradescope as usual.

FAQ

1. Can we use *<some other library>* in this PA instead of loops?

- Try to use the things we learned most recently. That said, if you know something you want to try, that's fine. Just be warned that it might be more work to not use the stuff we just learned

2. I wrote test methods with the Tester, but `./run` is telling me that no tests ran.

- Tester methods have to start with "test" at the beginning! For example, `boolean testAdd(Tester t) { …. }` . In `ArrayExamples.java` , all tests should be in class `ArrayExamples` , not `Pair` .

3. I failed some tests on Gradescope, but it is not showing me any error message, so I don't know what is wrong.

- We did something slightly different with this autograder. For each method we wrote several tests but only publicly show you some of their results in detail. The goal is to give you a nudge to think more about detailed tests you could write for your own methods. So you can see where you failed and all you know (for now) is that you should try testing and understanding those methods more.

4. My `earliest` method's tests on Gradescope are not passing.

- Double check your understanding of `compareTo` method. Try using `compareTo` with longer strings and seeing the result. Does the method always return 0, -1, or 1?

5. I am receiving an array index out of bounds error in `Longest.java` .

- Unlike some of the previous method, you can not make the assumption that string argument(s) will be given. Hence, args[0] will throw the index out of bounds error. What is a possible way to check if any arguments are given?

6. Help! I did the conditional checking and `Longest.java` is still throwing index out of bounds!

- Consider the following: `if (...){ // inside for loop } // after for loop` Recall that regardless if the if statement runs or not, the code beneath the if statement will run if not wrapped in an else statement. An else statement may be useful in avoiding the index out of bounds error.