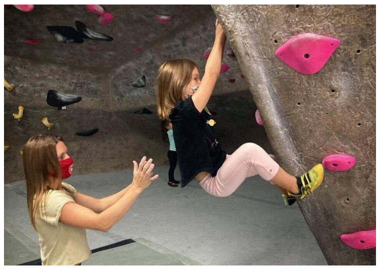
P01 Climbing Tracker

Overview

Over the past year, I've gotten into bouldering (a type of technical climbing done without a rope) at the local climbing gym, Boulders:



Pictured: not me but I think I did that route

Bouldering routes are given **grades** to reflect how difficult they are, ranging from V0 up to V15* (though you rarely see much over V7-8, so we'll top out there for the purposes of this program). It takes a lot of work and skill to move up a grade; I've been doing this for a year and a half and I'm only climbing around V3-4. So tracking progress is important!

For your first program, we'll be using Java to make a simple progress tracker for climbing, using arrays and some simple math functions.

* Boulders recently replaced these with color-coding but that's harder to do math on so we'll go with traditional grades for this

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on 09/13.
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 ordered 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 constants or variables outside of methods.
- 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 tests to verify the correctness of every public static method in your implementation.
- ALL test methods must be **public static**, take **zero** arguments, and return a **boolean** value. These methods MUST be contained in the ClimbingTrackerTester 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.
- Be careful when copying text directly from the spec; occasionally some characters will not be compatible with Java. If you run into compilation issues, check the error messages carefully.

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:

- Pair Programming Policy, which addresses such questions as:
  - How do I indicate I'm working with a partner?
  - What kinds of collaboration are allowed?
  - How do we submit a paired assignment?
  - and more!
- 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. Review the Pair Programming Policy and **REGISTER** your partnership before you begin. You MUST register your partnership by **11:59PM Sunday 09/12**.
2. Create a new project in Eclipse, called something like **P01 ClimbingTracker**.
   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.
3. Create two (2) Java source files within that project's src folder:
   a. ClimbingTracker.java (does NOT include a main method)
   b. ClimbingTrackerTester.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 **ClimbingTracker.java** program must contain the following methods. Implementation details, including required output formatting, are provided in later sections.

- `public static int sendClimb(String[] send, int numSend, String grade)`
    - Adds a successfully-completed ("sent") climb's grade to the end of the array of successful climbs if there is room AND the provided grade is valid (i.e., a capital `"V"` + a number 0-7).
    - Returns the resulting size of the `send` oversize array.
    - If there is no room to add a new successful climb OR the grade input is invalid, the return value will be the same as the initial value of `numSend`. Do not modify the array.
- `public static int failClimb(String[] fail, int numFail, String grade)`
    - Adds an unsuccessful climb's grade to the end of the array of failed climbs if there is room AND the provided grade is valid (i.e., a capital `"V"` + a number 0-7).
    - Returns the resulting size of the `fail` oversize array; if there is no room to add a new failed climb, the return value will be the same as the initial value of `numFail`.
    - **HINT**: since `sendClimb()` and `failClimb()` are very similar, this might be a good candidate for a helper method so you don't have to duplicate functionality.
- `public static String getStats(String[] send, int numSend, String[] fail, int numFail, int historyLength)`
    - Creates and returns a formatted String containing the **average** (mean) climb grade over the most recent `historyLength` number of climbs in each of the `send` and `fail` arrays. See pages 6 and 7 for formatting examples.
    - If `historyLength` is *greater* than either `numSend` or `numFail`, use the entire `send` or `fail` history.
    - If either `send` or `fail` are empty, or `historyLength` is 0 or negative, see page 7 for appropriate output.
- `public static String getHistogram(String[] send, int numSend, String[] fail, int numFail)`
    - Creates and returns a formatted String containing the number of climbs at each grade from V0 to the highest graded climb in either array. Failures are reported first, and are represented with a `"-"`; successes are represented with a `"+"` and are listed second. See page 7 for output examples.
    - If both arrays are empty, return an error message. See page 7 for an example.

Your **ClimbingTracker.java** class must NOT contain a main method. This class contains utility methods only, and will be run using the class described on the next page.

Your **ClimbingTrackerTester.java** program must contain AT LEAST the following methods.

- `public static boolean testSendClimb()`
  - Create an oversized array of Strings to test the `sendClimb()` method. Try adding to an empty array, a partially-filled array, a completely full array. Try adding valid and invalid grades. What return values do you expect from `sendClimb()` in each situation?
  - If you ever get a return value you do NOT expect from `sendClimb()`, return false. The test has failed.
  - If ALL of the return values match your expectations, return true. The test has passed.
- `public static boolean testFailClimb()`
  - Same idea as `testSendClimb()`.
- `public static boolean testGetStats()`
  - Create oversized arrays for both `send` and `fail`. Try multiple situations, including both typical cases (a mix of grades in both, multiple instances of one grade in both, etc) and edge cases (e.g. one or more empty arrays). What return values do you expect in these cases?
  - If you get return values that don't match your expectations, return false.
  - If ALL return values match your expectations, return true.
- `public static boolean testGetHistogram()`
  - Create oversized arrays for both `send` and `fail`. Try multiple situations, including both typical cases (a mix of grades in both, multiple instances of one grade in both, etc) and edge cases (e.g. one or more empty arrays). What return values do you expect in these cases?
  - If you get return values that don't match your expectations, return false.
  - If ALL return values match your expectations, return true.
- `public static boolean runAllTests()`
  - Contains one call to each test method you have written (at least four method calls; if you write more tests, call those here too).
  - Returns true if and only if all test methods return true; false otherwise.
- `public static void main(String[] args)`
  - Contains only one line: a method call to `runAllTests()`.

We recommend *beginning* your program with the **ClimbingTrackerTester.java** class! It's got the main method, so as you write your code, this class will allow you to run it and make sure it works.

You may add any printed output you like to your program. We will be testing only the return values.

Implementation Details and Suggestions

Begin your implementation ON PAPER: make sure you understand what outputs will be produced by various inputs. For example:

> If `send` and `fail` are empty 10-element arrays, `numSend` and `numFail` should be equal to?
>
> If I call `sendClimb(send, 2, "V0")` when `send` is a 10-element array, what does a correct implementation return?
>
> If I call `failClimb(fail, 10, "V0")` when `fail` is a 10-element array, what does a correct implementation return? Is it any different if I call `failClimb(fail, 10, "v0")`?

Once you've come up with enough of these that you think you've captured the requirements of the methods as laid out above, move to implementing ClimbingTrackerTester.java. The main method and runAllTests method are pretty simple, and the other test methods can be "stubbed" out as follows:

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

Start encoding the input/output combinations from your on-paper work in one of the test methods, so that if all the combinations are correct, the method will return true. Then FINALLY create that method in ClimbingTester.java and try to make it behave as you are expecting it to. Use your test method to see whether it works! Make sure you're commenting your code along the way ;)

Gradescope allows multiple submissions; once you have one of your test methods returning true on your implementation, submit to Gradescope and verify that you're passing our related tests, too.

Repeat until you're done!

Sample Output

As we said above, you may `System.out.println()` any output you like, but your String return values from `getStats()` should be formatted as follows (recall that to add a newline character to a string, just append `"\n"`):

```
send: {"V0", "V1", "V0", "V0", null}, numSend: 4
fail: {"V2", "V1", null, null, null}, numFail: 2

getStats(send, numSend, fail, numFail, 2):
      "send: 0.0
      fail: 1.5"
```

```
getStats(send, numSend, fail, numFail, 3):
        "send: 0.333333333
        fail: 1.5"

getStats(send, numSend, fail, 0, 3):
        "send: 0.333333333
        fail: --"

getStats(send, numSend, fail, numFail, 0):
        "send: --
        fail: --"
```

Don't worry about the number of significant digits in the output; just report the value as calculated. Note that you will need to remove the leading `"V"` from the grade in order to calculate the average.

If one or both of the `send/fail` arrays is empty, do not calculate the average (since that will have you dividing by zero and That's Bad); just display two dash characters (`"--"`) instead of the average.

For `getHistogram()`, your output should be formatted as follows:

```
send: {"V0", "V1", "V0", "V0", null}, numSend: 4
fail: {"V2", "V1", null, null, null}, numFail: 2

getHistogram(send, numSend, fail, numFail):
        "V0: + + +
        V1: - +
        V2: -"

getHistogram(send, 0, fail, 0):
        "Error: no data to display"
```

Note that all successes (`"+"`) are reported AFTER any failures (`"-"`) since we do not preserve any ordering between success and failure. These characters should be separated by **spaces**. You may include (or not) a trailing space on each line; we will allow either interpretation in our testing.

If there are NO attempts for a given grade but there are attempts for a higher one, your output will look like this:

```
send: {"V0", "V1", "V0", "V0", null}, numSend: 4
fail: {"V3", "V1", null, null, null}, numFail: 2

getHistogram(send, numSend, fail, numFail):
        "V0: + + +
        V1: - +
        V2:
        V3: -"
```

You may include (or not) the space following the colon after a grade with no attempts; we will allow either interpretation in our testing.

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.

If you chose to work with a partner on this assignment, you are responsible for submitting all proper documentation and ensuring your code has been submitted properly.

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

- ClimbingTracker.java
- ClimbingTrackerTester.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