


CSE11-SP21-Assignments / cse11-sp21-pa7-starter

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)main [cse11-sp21-pa7-starter / README.md](#)[Go to file](#)

...

 shihualu Update README.mdLatest commit a9fe664 14 hours ago [History](#) 1 contributor

189 lines (150 sloc) | 12.3 KB

[Raw](#) [Blame](#)

CSE-11 Programming Assignment 7

Due Date: Wednesday, May 19, 11:59PM Pacific Time

Learning Goals

In this assignment you will put together many of the programming features and techniques you've seen so far to build a useful command-line application for text manipulation.

This program is larger than the ones you've written so far for this course, so you should **start early** so you understand the scope. Use the implementation suggestions and milestones to help you get started and work incrementally.

Submission checklist:

- ☐ StringSearchMilestone1.java
- ☐ StringSearchMilestone2.java
- ☐ StringSearchMilestone3.java
- ☐ StringSearchMilestone4.java
- ☐ StringSearch.java

In addition to correctness, your code will also be graded on styles this time. Check this [Piazza post](#) and this [Piazza post](#) for some coding style guidelines.

Collaboration

Different assignments in this course have different collaboration policies. On this assignment, you cannot share code publicly on Piazza or with anyone other than the course staff. If you need to share code, ask in a private Piazza post or in 1-on-1 hours. Still do ask any public code questions about lecture code, quizzes that are past, or other conceptual questions, just no code from this PA. You can't get help from anyone but the course staff on this PA.

On this assignment, we **encourage** you to share publicly and with other students what you think the expected output should be on particular examples. For example, you could share a `java StringSearch ...` command you tried out, and show the results, and check with other students if they agree on the behavior. This allows you to discuss how the assignment is supposed to work without sharing any code, and you also might want to share examples you found interesting!

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 resubmission assignment in Gradescope to earn a maximum of 75% the points.
- If you scored higher than 75% on the original 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.

Task

In a file called `StringSearch.java`, you'll write a class `StringSearch` with a `main` method that uses command-line arguments as described below. You can write as many additional methods and classes as you wish, and use any Java features you like. We have some suggestions in the program structure section later on that you can use, or not use, as you see fit.

The `main` method should expect 3 command-line arguments:

```
$ java StringSearch "<file>" "<query>" "<transform>"
```

The overall goal of `StringSearch` is to take a file of text, search for lines in the file based on some criteria, then print out the matching lines after transforming them somehow.

The `<thing>` syntax means, as usual, that we will be describing what kinds of syntax can go in each position in more detail.

- `<file>` should be a path to a file. We've included two for you to test on with examples below. You should make a few of your own files and try them out, as well.
- `<query>` describes criteria for which lines in the file to print.
- `<transform>` describes how to change each line in the file before printing.

If just a file is provided, the program should print the file's entire contents, and if just a file and a query are provided with no transform, the program should just print the matching lines (see examples below).

Queries

The `<query>` part of the command-line should be a `&`-separated sequence of individual queries. The individual queries are:

- `length=<number>` which matches lines with exactly `<number>` characters
 - `greater=<number>` which matches lines with more than `<number>` characters
 - `less=<number>` which matches lines with less than `<number>` characters
 - `contains=<string>` which matches lines containing the `<string>` (case-sensitive)
 - `starts=<string>` which matches lines starting with the `<string>`
 - `ends=<string>` which matches lines ending with the `<string>`
 - `not(<some non-not individual query>)` which matches lines that do not match the inner query
- "length = 10 & contains = 'hi'"*

Transforms

The `<transform>` part of the command-line should be a `&`-separated sequence of individual transforms. The individual transforms are:

- `upper` which transforms the line to uppercase
- `lower` which transforms the line to lowercase
- `first=<number>` which transforms the line by taking the first `<number>` characters of the line. If there are fewer than `<number>` characters, produces the whole line
- `last=<number>` which transforms the line by taking the last `<number>` characters of the line. If there are fewer than `<number>` characters, produces the whole line
- `replace=<string>;<string>` which transforms the line by replacing all appearances of the first string with the second (some lines might have no replacements, and won't be transformed by this transform)

If there is a sequence of `&`-separated transforms, the transforms should be applied in order, from left to right.

Where you see `<string>` above, it should always be characters inside single quotes, like `'abc'`. We chose this because it works best with command-line tools.

Where you see `<number>` above, it should always be a positive integer. *Integer.parseInt(c) String.valueOf(c)*

The `<file>`, `<query>`, and `<transform>` command-line arguments should always be inside double quotes. This ensures that they won't be interpreted as commands, or parts of commands, by your terminal.

Examples

The file `poem.txt` contains the following content:

This is a short file ✓
It contains text and this is ✓
Also a haiku

The file `words` contains a standard dictionary.

The following commands, when run at the command line, should produce the given outputs.

```
$ java StringSearch "poem.txt"
This is a short file
It contains text and this is
Also a haiku
$ java StringSearch "poem.txt" "greater=13"
This is a short file
It contains text and this is
$ java StringSearch "poem.txt" "not(contains='short')"
It contains text and this is
Also a haiku
$ java StringSearch "poem.txt" "greater=13&starts='This'"
This is a short file
$ java StringSearch "poem.txt" "contains='his'" "last=10"
short file
nd this is
$ java StringSearch "poem.txt" "contains=' a '" "upper&first=18"
THIS IS A SHORT FI
ALSO A HAIKU
$ java StringSearch "poem.txt" "greater=3&less=100&not(ends='z')" "replace='i';'I'"
ThIs Is a short fIle
It contaIns text and thIs Is
Also a haIku
$ java StringSearch "poem.txt" "greater=3&less=100&not(ends='u')" "replace='i';'I'"
ThIs Is a short fIle
It contaIns text and thIs Is
$ java StringSearch "words" "contains='no'&starts='x'&not(contains='xeno')" "lower"
xanthocyanopsia
xanthocyanopsy
xanthocyanopy
xanthomelanous
xoanon
xylenol
xyloquinone
xylorcinol
```

Milestones

You **must** submit the following milestones with your submission. You should save them in files called `StringSearchMilestone1.java`, `StringSearchMilestone2.java`, and so on. This is the main way we will grant partial credit, and they also serve to help you break the program into small chunks of progress.

We recommend working on the first milestone directly in `StringSearch.java`. Once it's working, you can save to the `StringSearchMilestone1.java` file to record your progress and keep working in `StringSearch.java`. You can then work towards the second, copy your work over when done with that milestone, and so on to build up your submission. Note that for your milestones to compile successfully on Gradescope, the class name should just be `StringSearch` rather than `StringSearchMilestone1/2/3/4`.

You can upload any of the milestones to Gradescope to get grading feedback on them. If you finish multiple milestones at once, you can always copy your more advanced solution as the earlier milestone files – you're only adding, not changing features as you go through the milestones.

Milestone 1

Your program should take in one argument that is the name of the file to read, and print out all the lines in that file in order.

Milestone 2

Your program should take in the name of the file and a *single* `contains` query, and print all the lines that match that `contains` query.

Milestone 3

Your program should take in the name of the file and a *single* query of any type, and print all the lines that match that single query. By *single*, we mean that any single query (length, greater, less, contains, starts, ends, not) could be used to test the Milestone3 code, but not a sequence of queries separated by `&`.

Milestone 4

Your program should take in the name of the file and a *single* query of any type and a *single* transform of any type, and print all the lines that match that single query, transformed by that single transform.

After milestone 4, complete the full task as described above.

Suggestions

Implementation Ideas

You may find it useful to write or use some of the following components while implementing your program. These are not required, and you may find only some of them useful to you, or you might find all of them useful to you. We do think they are all useful, so we aren't trying to trick you or anything 😊. If you do choose some other approach, during help hours we may guide you towards these if we think your choices won't lead to a successful design.

- An interface `Query` that has a `boolean matches(String s)` method
- Several classes representing different individual queries like `Contains`, etc
- A method `Query readQuery(String q)` that takes a query string and produces a `Query`
- An interface `Transform` that has a `String transform(String s)` method
- Several classes representing different individual transforms like `ToUpper`, etc
- A method `Transform readTransform(String t)` that takes a transform string for a single transform and produces a `Transform`
- The built-in Java `String split` method `String s, String sep → String[]`
- A method `boolean matchesAll(Query[] qs, String s)` that returns `true` if all the queries match a given string.
- A method `String applyAll(Transform[] ts, String s)` that returns the result of applying all of the `Transform`s in `ts` to `s` in order.

Class String Search

Reading Files

Part of this PA requires that you read in the contents of a file. You can include this code snippet at the top of your file:

```
import java.nio.file.*;
import java.io.IOException;

class FileHelper {
    /*
```

`s = "a & b & c" "&"`
`["a", "b", "c"]`

Takes a path to a file and returns all of the lines in the file as an array of strings, printing an error if it failed.

```
*/
static String[] getLines(String path) {
    try {
        return Files.readAllLines(Paths.get(path)).toArray(String[]::new);
    }
    catch(IOException e) {
        System.err.println("Error reading file " + path + ": " + e);
        return new String[]{"Error reading file " + path + ": " + e};
    }
}
```

Submission

Submit your 5 code files (4 milestones and final `StringSearch.java`) to `pa7` on Gradescope. There will be an autograder available while the assignment is out, and we will also give you feedback on your code after submission. If your queries and/or transforms worked fine locally but failed on the autograder, this is because the autograder checks for some more interesting cases other than the sample tests provided in the writeup. Try to come up with more thorough test cases when testing locally!

Extensions

These are not for credit, but you may find them interesting to try on your own.

1. Add a new transform of your own design.
2. Add a new query of your own design.
3. All of the queries above are joined in the style of `AndQuery` from the `ImageQuery` reading, where they all need to be true to match a line. Extend your implementation to accept both `&` and `|` as separators between queries, where `|` indicates joining queries by **or** rather than **and**. Describe the design and details of how to read in and interpret a mix of `&` and `|` within a query to a user.