Homework 03 - File Reader

Authors: Emma Barron, Ryan Miles, Tushna Eduljee, Manny Sonubi, Allan Nguyen, Shishir Bhat

Problem Description

Your company has a text file full of commands they don't know how to execute. Now that they've hired you, they want you to do it! Write a program that takes in a text file via command line. This text file has a list of commands. Write a Java class, `FileReader`, that reads the file, follows the commands, and prints out the results.

Solution Description

The text file will need to be in the same directory as your `FileReader.java` file.

There are three possible commands in the text file. Each command will be followed by its arguments separated by spaces. There will always be the correct number of arguments. The arguments will always be the correct type (e.g. you will only get numbers for `power`, we will not give you a string like "hi"). The commands are `allcaps`, `power`, and `substring`.

- `allcaps` has one argument, a string that you will convert to uppercase. For example, the result of `allcaps Hello` would be `HELLO`. The string will always be one word.
- `power` has two arguments, a base and a power to raise that base to. For example, the result of `power 2 3` would be `8.0`
- `substring` has three arguments, a string, a start index, and an end index. For example, `substring cs1331rocks 6 11` would result in `rocks`. The string will always be one word. The start index is inclusive, the end index exclusive. If either the start or end index is out of bounds, the result should be "Invalid command" instead of the substring.

You are encouraged to use methods in the Java String class and the Java Math class to help you capitalize strings, calculate exponents, and create substrings.

For this assignment, you won't be expected to write your own class. We have provided a `FileReader.java` file. In the main method, there will be a variable called `commandsFile`. You will use that variable to read from the file.

You will add three static methods, one for each command.

```
public static String allCaps(String str)
```

```
public static double power(int base, int power)
```

```
public static String makeSubstring(String str, int startIndex, int endIndex)
```

You will perform your computations in the method, then print the results from the main method. **Again, don't print anything from these methods, just from `main`.**

Testing your app

In order to test your program, you must compile by running `javac FileReader.java` in the same folder as `FileReader.java`

Then you can run `java FileReader [fileName]`, providing some example file. The example file should be in the same directory as your program.

For example, if `exampleFile.txt` is

```
allcaps Hello
power 2 3
substring cs1331rocks 6 11
substring hi 0 3
```

running `java FileReader exampleFile.txt` prints out

```
HELLO
8.0
rocks
Invalid command
```

Remember that what you print out should match the defined output *exactly*. Deviation from what the output should be, even by a character or two, will result in lost points.

Rubric

- [15] correct printed output
- allcaps
  - [5] correct method signature & return type
  - [20] correct method output
- power
  - [5] correct method signature & return type
  - [20] correct method output
- makeSubstring
  - [5] correct method signature & return type
  - [20] correct method output
  - [10] handles invalid start and end indices

Allowed Imports

To prevent trivialization of the assignment, you are *only* allowed to import `java.util.Scanner`, `java.io.File`, and `java.io.FileNotFoundException`.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- var (the reserved keyword)
- System.exit

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error. For this homework the checkstyle cap is 10, meaning you can lose a maximum of 10 points on this assignment due to style errors. This limit will increase with each homework.

If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can! You can run checkstyle on your code by using the jar file found on Canvas (under Files/Resources). To

check the style of your code, place the checkstyle jar file in the same folder as your java files, then run java -jar checkstyle-6.2.2.jar *.java. You will be responsible for running checkstyle on ALL of your code.

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- FileReader.java

Make sure you see the message stating "HW03 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications