HW8 simplified grep

Problem Description

Grep is a command line utility for finding regular expessions (strings) in a directory hierarchy. An acronym which stands for "**g**lobally search a **r**egular **e**xpression and **p**rint," grep is one of the most useful utilities avaliable on Unix operating systems (Mac OS and Linux).

You will be implementing a Java version of grep. Just like the real grep, it must be useable as a command line utility. It must be able to search a user-specified directory hierarchy for a user-specified String. Your grep should support both case-sensitive and case-insensitive searching. It should print to the console the specific line(s) of every file which contain the search string, as well as the specific location of the line (file and line number), as well as handle excepions appropriately. The grep you write will have the same functionality as running `grep -rn` or `grep -rni` on a Unix machine.

If you're interested in more details about the real grep, see the Linux grep Manual and this webpage which walks through some common uses.

Solution Description

Create a runnable java class named Grep that will help a person find all the occurrences of a string in any files within a directory (folder), as well as in any of its subdirectories (subfolders).

As arguments on the command line, the program must take in an optional `-i` flag (to search case-insensitively), the string to search for, and the directory to be run on. The directory specified must be searched recursively. This means all files in the directory, all its subdirectories, all the subdirectories of the subdirectories, etc, all the way down. The program may be run on the directory "/books" with the search string "never again" as follows:

```
java Grep "never again" books
```

or

```
java Grep -i "never again" books
```

The arguments will always be in the order shown.

When run, it will search all files in the given directory and subdirectories. On any line where the search string is found print out the **whole line containing the match** that contains the search string in the format `[filename]:[linenumber]:[line containing match]`, with each match on its own line.

We have provided a couple of directories with text files to test on. For example, you man run the program on the provided submissions directory:

```
java Grep "I worked on the homework assignment alone, using only course materials." sampleHW04/submissio
```

is run,

```
sampleHW04/submissions/aalice256/hw08Submission/Cat.java:1://I worked on the homework assignment alone,
sampleHW04/submissions/bbob42/Cat.java:1://I worked on the homework assignment alone, using only course
```

Should be printed. Other test:

Command:

```
java Grep -i "I worked on the homework assignment alone, using only course materials." sampleHW04/submis
```

Output:

```
submissions/hlife3/Dog.java:1://i worked on the homework assignment alone, using ONLY course materials.
submissions/aalice25/Cat.java:1://I worked on the homework assignment alone, using only course material
submissions/eeve133/Dog.java:1://i worked on the homework assignment ALONE, using only course materials
submissions/bbob42/Cat.java:1://I worked on the homework assignment alone, using only course materials.
```

Note: the extra matches due to searching case-insensitively.

Command:

```
java Grep "never again" books
```

Output:

```
books/A/LittleWomen.txt:10289:fever was a thing of the past.  Not an invalid exactly, but never again
books/A/Austen/PrideAndPrejudice.txt:10948:never again distressed himself, or provoked his dear sister
books/A/Austen/Emma.txt:12654:Bates should never again--no, never! If attention, in future, could do
books/R/TheHistoryOfJava.txt:17344:his power) that he cursed them, and swore that they should never aga
books/R/TheHistoryOfJava.txt:18427:Búmi_, that the earth might never again see to attack the heavens, a
books/D/TheCountOfMonteCristo.txt:9269:with nothingness! Alone!-never again to see the face, never agai
books/D/TheCountOfMonteCristo.txt:14950:the poor fellow never again set foot in Tunis. This was a usele
books/D/TheCountOfMonteCristo.txt:24829:and the money stolen. Benedetto never again appeared at Roglian
books/D/TheCountOfMonteCristo.txt:53259:and she should never again see Morrel!
books/D/TheCountOfMonteCristo.txt:60437:Happily, I vanquished death. Henceforth, Valentine, you will ne
books/B/Bronte/Villette.txt:1508:departure--little thinking then I was never again to visit it; never
books/B/Bronte/Villette.txt:13725:enlightened him, and taught him well never again to expect of me the
books/B/Bronte/JaneEyre.txt:11518:"I will never again come to your side: I am torn away now, and cannot
books/B/PeterPan.txt:3672:tree. As so often before, but never again.
books/B/PeterPan.txt:4487:personification of cockiness as, taken together, will never again, one
books/M/MobyDick.txt:9832:probability would be that he and his shipmates would never again
books/M/MobyDick.txt:19748:But in either case, the needle never again, of itself, recovers the
books/S/Dracula.txt:2473:never again think that a man must be playful always, and never earnest,
books/J/Ulysses.txt:17604:him aside as if he was so much filth and never again would she cast as
books/J/Ulysses.txt:32116:seaside but Id never again in this life get into a boat with him after
```

If a non-existent directory is given or any other IOExceptions happen, the program must simply print a user friendly error message and quit.

You are required to write two methods in the Grep class: a `main` method, and a `grep` method. You may write other helper methods if you would like; however, be sure that the required methods throw or catch exceptions as specified. You are also required to **write a custom exception class** InvalidSearchStringException for use in the other methods.

1. Main method:

Write a standard `main` method as you have been taught. It should parse the command line arguments and call `grep` with correct arguments. It should print out the result of `grep`.

If the incorrect number of command line arguments are given, `main` should throw an IllegalArgumentException, a class in Java's standard library. It should also catch any exceptions thrown by `grep` and print an error message explaining the error.

2. Grep methods:

The parameters for `grep` should be a File object, a String, and a boolean for case-sensitively. If the boolean is true, search case-sensitively, otherwise, search case-insensitively. Make sure to reuse code where possible. `grep` should be static.

The File class in Java may represent either a directory or a file. The parameter passed to the `grep` method will always represent a directory in any tests used in grading.

`grep` must read any files contained in the directory, and search each file for the given String. We recommend using a Scanner to read from a file, but there are multiple classes in the API that can do this.

It must also execute recursively on any sub-directories contained in the given directory. The `grep` method doesn't necessarily have to call itself directly, as long as it calls a method that is recursive, i.e. it may call a recursive helper method or a helper method that calls `grep` again.

`grep` must return a String containing all of the matches from all of the files, both in the directory and subdirectories. The order of the matches in the String doesn't matter.

If the String parameter of `grep` contains a new line character, an InvalidSearchStringException must be thrown.

These methods should rely on their callers to handle any IOExceptions.

3. InvalidSearchStringException Class:

This class should be a runtime exception to be used by `grep`. You may give it any fields you see fit.

Tips and Considerations

- This assignment is short, but the logic is difficult. Make sure to start early!

Allowed Imports

You are allowed to use any of file IO or managing classes in the Java Standard Library. However, if you choose to use anything other than that mentioned in the assignment, the TA's may be unable to help you in office hours.

You are not allowed to use any of the collections framework in java, including ArrayList and Streams.

Feature Restrictions

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

- var (the reserved keyword)
- System.arraycopy

Rubric

- [20] **Main Method**
  - [5] Parses arguments correctly
  - [5] Throws IllegalArgumentException when given incorrect arguments
  - [5] Calls `grep` and prints results
  - [5] Catches and handles exceptions thrown by `grep`
- [60] **Grep Method**
  - [5] Static `grep` method with correct return type and parameters
  - [10] Reads files in the given directory
  - [10] Correctly searches files in given directory
  - [10] Recursively calls itself on sub-directories

    - [10] Output string is formatted correctly
    - [5] Throws InvalidSearchStringException when given invalid string
    - [5] Relies on caller to handle IOException
    - [5] Doesn't significantly duplicate code
- [20] **InvalidSearchStringException Class**
    - [10] Is an exception
    - [10] Is a runtime exception

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```java
import java.util.Scanner;

/**
 * This class represents a Coffee object.
 * @author George P. Burdell
 * @version 1.0
 */
public class Coffee {

    /**
     * Creates a new Coffee
     */
    public Coffee() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b) {
    ...
    }

}
```

A more thorough tutorial for Javadocs can be found here.

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.

2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section. Just like Checkstyle, **one point will be deducted for each Javadoc error detected by running checkstyle**. These deductions will also be limited by the checkstyle cap, which applies to the sum of checkstyle and javadoc errors.

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points. Review the style guide and download the checkstyle jar. Run checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar -a *.java
Audit done. Errors (potential points off):
0
```

Make sure to include the -a flag included for testing both checkstyle and javadocs

The message above means there were no Checkstyle or javadoc errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips page for more information.

Collaboration

Collaboration Statement

To ensure that you acknowledge a 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].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Yo it's 10:40 on Thursday… Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload all java files you have created from this homework.

Make sure you see the message stating "HW08 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 are also autograded once the submission deadline has passed. The autograders used are often dependent on specific output formats, so **make sure that your submission passes all test cases marked "FORMAT:"**.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Ensure you pass all "FORMAT:" tests
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications