CMPS 12M-02
Data Structures Lab
Lab Assignment 3 (Lab3)
Due: Monday, October 23 by 11:59pm

- **Assignments submitted late will lose 1 point if submitted less than one day (24 hours) late.**
- **Assignments submitted more than one day late will receive no credit.**

The goal of this assignment is to practice using command line arguments, file input/output, and manipulation of Strings in java. File input/output and command line arguments is essential for assignments. Some students have had difficulty with HW1 because they were not used to input/output.

Command Line Arguments

A java main function always reads the operating system command line from which it was called, and stores the tokens on that line in the `args` array. Use the following Java program to create an executable `jar` file called `CommandLineArguments` (see Lab2 to learn how to do this)

```
// CommandLineArguments.java
class CommandLineArguments{
   public static void main(String[] args){
      int n = args.length;
      System.out.println("args.length = " + n);
      for(int i=0; i<n; i++) System.out.println(args[i]);
   }
}
```

then run `java -jar CommandLineArguments.jar zero one two three four` and observe the output. Run it with several other sets of tokens on the command line. These tokens are called command line arguments, and can be used within a program to specify and modify the program's behavior. Typically command line arguments will be either strings specifying optional behavior, text to be processed by the program directly, or names of files to be processed in some way.

File Input/Output

The java.util package contains the Scanner class, and the java.io package contains classes PrintWriter and FileWriter. These classes perform simple input and output operations on text files. Their usage is illustrated in the program `FileCopy.java` below, which merely copies one file to another, i.e. it provides essentially the same functionality as the Unix command `cp` (with respect to text files only.)

```
// FileCopy.java
// Illustrates file IO

import java.io.*;
import java.util.Scanner;

class FileCopy{

   public static void main(String[] args) throws IOException{

      // check number of command line arguments is at least 2
      if(args.length < 2){
         System.out.println("Usage: java –jar FileCopy.jar <input file> <output 
file>");
         System.exit(1);
      }

      // open files
      Scanner in = new Scanner(new File(args[0]));
      PrintWriter out = new PrintWriter(new FileWriter(args[1]));

      // read lines from in, write lines to out
      while( in.hasNextLine() ){
         String line = in.nextLine();
         out.println( line );
      }

      // close files
      in.close();
      out.close();
   }
}
```

As you can see, the Scanner constructor takes a File object for initialization, which is itself initialized by a String giving the name of an input file. The Scanner class contains (among others) methods called `hasNextLine()` and `nextLine()`. Read the documentation for Scanner at:

http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

to learn about the proper usage of these methods. The PrintWriter constructor takes a FileWriter object for initialization, which is in turn initialized by a String giving the name of an output file. PrintWriter contains methods `print()` and `println()`. When you use `System.out.println()` you are calling the `println()` method for an instance of `PrintWriter` that was initialized to write to `stdout`. Note that the FileWriter initialization can fail if no file named `args[1]` exists in the current directory. If it fails, it will throw an IOException. This is a checked exception, which cannot be ignored, and therefore function `main()` must either catch the exception, or throw it up the chain of function calls. (In the case of function `main()`, the "calling function" is the operating system). In this example, we deal with this by declaring `main` to throw an IOException, causing the program to quit if the exception is encountered. Similar comments apply to the initialization of the Scanner object. See the java documentation for more details.

Compile and run `FileCopy.java`, and observe that a `Usage` statement is printed if the user does not provide at least two command line arguments. This `Usage` statement assumes that the program is being run from an executable jar file called `FileCopy`. All of your programs that take command line arguments should include such a usage statement. If the usage statement was printed when you ran `FileCopy.java`, try running it again with a file and observe the behavior by viewing each file after it has ran.

String Tokenization

A common task in text processing is to parse a string by deleting the surrounding whitespace characters, keeping just the discrete words or “tokens” which remain. A token is a maximal substring containing no whitespace characters. For instance, consider the preceding sentence to be a string. The 10 tokens in this string are: "A", "token", "is", "a", "maximal", "substring", "containing", "no", "whitespace", "characters.". Whitespace here is defined to mean spaces, newlines, and tab characters. This is one of the first tasks that a compiler for any language such as Java or C must perform. The source file is broken up into tokens, each of which is then classified as: keyword, identifier, punctuation, etc. Java's String class contains a method called `split()` which decomposes a string into tokens, then returns a String array containing the tokens as its elements. Compile and run the following program `FileTokens.java` illustrating these operations.

```
//-----------------------------------------------------------------------------
// FileTokens.java
// Illustrates file IO and tokenization of strings.
//-----------------------------------------------------------------------------
import java.io.*;
import java.util.Scanner;

class FileTokens{
   public static void main(String[] args) throws IOException{

      int lineNumber = 0;

      // check number of command line arguments is at least 2
      if(args.length < 2){
         System.out.println("Usage:  java  –jar  FileTokens.jar  <input  file> 
<output file>");
         System.exit(1);
      }

      // open files
      Scanner in = new Scanner(new File(args[0]));
      PrintWriter out = new PrintWriter(new FileWriter(args[1]));

      // read lines from in, extract and print tokens from each line
      while( in.hasNextLine() ){
         lineNumber++;

         // trim leading and trailing spaces, then add one trailing space so
         // split works on blank lines
         String line = in.nextLine().trim() + " ";

         // split line around white space
         String[] token = line.split("\\s+");

         // print out tokens
         int n = token.length;
         out.println("Line " + lineNumber + " contains " + n + " tokens:");
         for(int i=0; i<n; i++){
            out.println("  "+token[i]);
         }
      }
```

```
        // close files
        in.close();
        out.close();
    }
}
```

Reversing a file

Write a java program called `FileReverse.java` that takes two command line arguments giving the names of the input and output files respectively (as shown in the preceding examples). Your program will read each line of input, parse the tokens, then print each token backwards to the output file on a line by itself. For example, you might be given a file called `in` containing the lines:

```
abc defg
hi
jkl mnop q

rstu v
wxyz
```

Write a Makefile to create a jar file `FileReverse.jar`. The command:

```
      java -jar FileReverse.jar in out
```

should create a file called `out` containing the lines:

```
cba
gfed
ih
lkj
ponm
q
utsr
v
zyxw
```

Your program will contain a method called `stringReverse()` with the following signature:

```
      public static String stringReverse(String s)
```

This function will return a String that is the reversal of `s`. Note that reversing a String is very similar to reversing an array. Study the methods in the String class documented at:

http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

to determine how this might be done. See especially the instance methods `charAt()` and `substring()`, as well as the static method `valueOf()`. Use `stringReverse()` to perform the reversal of tokens from the input file.

You may write stringReverse with or without using recursion, as you prefer. Chapter 3 of P&C has a recursive method to write a string backward; since it's in the textbook, you may use it without penalty.

What to submit:

On Piazza under Lab3, there's a file called Lab3-files.zip; if you unzip that, you'll get the spec file for Lab3, `Lab3-spec.txt`, as well as two files, `gettysburg.txt` and `gettysburg-reversed.txt`, that may be used for testing.

Create a folder Lab3 in your git repo to submit your Lab3 files. Submit the files `FileReverse.java`, `Makefile`, `README`. You will also need to put test files `gettysburg.txt` and `gettysburg-reversed.txt` into your Lab3 folder for the checking script to work for `Lab3-spec.txt`, the provided spec-file. You may (but you don't have to) submit the 2 gettysburg files through git.

Be sure to add any online resources you've used or descriptions of discussions you've had with others in the README file. (You must submit a README file, even if you haven't had any help.) Please run the checking script (either version that's on Piazza) to ensure that you pass. Piazza has guides on submitting your assignments and for running the checking script.

Git commit id:

As for Lab1 and Lab2, you need to submit your git commit id in a Google Form, but there's a different Google Form for Lab3 (and for every new Lab/HW). Follow the directions in Lab1, Part 3 to get the git commit id after you pushed your files into the repo and typed "git log". Then enter your 40-character commit id into the following (new) Google Form:

https://docs.google.com/a/ucsc.edu/forms/d/1a2wQzExqOBM2t8fDHxgtw7xN5NDS8rX2MtwAon2aneo/

Note that this is a required part of the assignment, as the Grading described below shows.

Grading:

- (10 points) Everything has been done correctly.
- (6 points) FileReverse.java fails on some inputs.
- (0 points) Git Commit id has not been submitted.
- (0 points) Checking script fails.

As usual, please start early and ask questions in lab section or office hours if anything is unclear.