CS1130 Lab 05. Reading files Spring 2012

Name _____ NetId _____

This lab discusses input —reading a file. After the lab, study section 5.9 of the text. Better yet, listen to the lectures on lesson page 5-7 of the ProgramLive CD. The lectures are much clearer than the paper version.

Start this lab by downloading file lab05.zip putting its contents in a new directory. It contains these files: Lab05.java , cms.txt, and peop.txt. Then open the .java file in DrJava and compile.

Streams

A "stream" is a sequence of data values that is processed —either read or written— from beginning to end. When the data is being read, or input, the stream is called an "input stream"; when it is being written, or output, the stream is called an "output stream". Input/output of streams is done in Java using classes in package java.io.

The basic way to create an input stream for a file is by creating an instance of class `FileReader`:

```
FileReader fr= new FileReader(an arg that describes what file to read);
```

And the standard way to read using `FileReader fr` is to read one character at a time, using function

```
fr.read()
```

This is too low-level for us. We would like to be able to read not one character at a time but one line at a time. For this, we use class `BufferedReader`. Instead of the above, use this:

```
FileReader fr= new FileReader(an arg that describes what file to read);
BufferedReader br= new BufferedReader(fr);
```

Then, execution of

```
String lin= br.readLine();
```

reads the next line of the file and stores it in variable `lin` —if there are no more lines to read, `null` is stored in `lin`. We will see how to use this later.

Using a JFileChooser dialog box

In order to read a file, you have to indicate which file should be read. The easiest way to do this is to use a dialog window to navigate to the appropriate directory and select the file. For this, we use an instance of class `JFileChooser`, in package `javax.swing`. Execute the following in the Interactions pane:

```
br= Lab10.getReader(null);
```

A dialog box opens. Its title is "Choose input file". And it allows you to navigate anywhere you want and then select a file. Do a bit of navigating and select a file. Then take a look at function `getReader(String)`. Here is what it does:

1. Declare local variable `jd` of class `JFileChooser`.
2. Store in `jd` a new instance of class `JFileChooser`. In the new expression **`new`** `JFileChooser()`, you have no control over the directory that appears in the dialog window initially. In the new expression **`new`** `JFileChooser(p)`, String `p` is supposed to be a path on your computer of the directory to open in the dialog window. This choice allows you to dispense with a lot of navigating.
3. Set the title of `jd` to "Choose input file".
4. Execution of `jd.showOpenDialog(null);` causes the dialog window to open on your monitor,

and the program pauses until you have closed it. Nothing happens until you have chosen a file (or canceled the interaction).

5. Create a new `FileReader` with the file that you selected (its name is `jd.getSelectedFile()`) as the argument and store its name in `fr`.
6. Create and return a `BufferedReader` that is attached to `fr`.

The function for obtaining the next line from a `BufferedReader br` is:

```
br.readLine()    // = the next line of BufferedReader br —null if there are no more lines
```

In the Interactions pane, you can continue to evaluate `br.readLine()`. Each time you do it, the next line of the file you selected is printed. Try it.

Processing the lines of a file

Function `lines` in class `Lab10` illustrates the basic way of processing the lines of a file given by a `BufferedReader`. In the Interactions pane, put a call on this method and let it read some file —you will see how many lines the file has in it.

Study this method. Any loop that you write that processes a file should be similar to this one —the "processing" of each line will change, but the basic structure of the loop that does the processing will not. Note that this method contains a while-loop. Here are important points:

1. The first line is read before the while loop and stored in variable `lin`. `lin` will be `null` if the file is empty.
2. The loop stops when `lin` is `null`, indicating that there are no more lines in the file.
3. The repetend first processes the line given by variable `lin` and then reads the next line into `lin`.

The header of the method contains a new construct: `throws IOException`. It is needed because function `br.readLine()` might create some sort of I/O (input/output) error, and this is how we handle it. We will explain this later in the course.

Write your own method

Write the body of function `contains` that is declared in class `Lab10` (and test it). (If you are smart, you will write a JUnit testing class with a procedure to test this method.) It can be used, for example, to see whether a Cornell netid likd djg17 appears in one of the two files cms.txt and peop.txt. Show your function to the Lab instructor when you are finished.

Writing files

This section describes a task that you might have fun doing. It is something we had to do in preparing for prelim 2, because of a huge number of conflicts with a math course. The task is to read in two files of netids, say cms.txt and peop.txt (containing netids for our course and netids for a particular math course), and create a file that contains all netids that are in both files. You can assume that the two files are already sorted by netids. This is nice example works with input/output files and also contains an interesting loop ---which can be simple or comlpicated, depending how YOU write it.

Please read on and learn about writing files, but you do not have to complete this task for this lab.

Writing files is not much different from reading them. Procedure `getWriter` obtains the name of a file (in a particular directory) to write, using a `JFileChooser` window that is placed on the monitor using `showSaveDialog` (instead of `showOpenDialog`). The procedure uses the resulting file as an argument in the constructor call of a `PrintStream` object.

Try this in the Interactions pane: `v= Lab10.getWriter();`

When the dialog window opens, navigate to a directory and select a file name or type in a completely

different one ---you can select one and then change it slightly. BE CAREFUL. You will not want to overwrite an existing file, but this can easily happen.

Now, type

```
v.println("first line in the file");
v.close();
```

The first statement writes a line to the file. You can write as many lines as you wish. The second statement should be used to close the file.

After these statements are executed, look at the directory where the file was written, open the file, and look at it.

Now, write (and test) the body of procedure `Lab10.extract`. Note that it has a series of comment-statements ---each is a statement to do something, written in English. Write the Java implementation of each comment directly under the comment and follow the implementation with a blank line.