

## CS100J Lab 09. Reading files Spring 2006

Name \_\_\_\_\_ Section time \_\_\_\_\_ Section instructor \_\_\_\_\_

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 files [Lab09.java](#) into a directory, opening them in DrJava, and compiling. You will also need this text file: [test.txt](#).

### 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". All 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 argument that describes which file to read);
```

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

```
fr.read()
```

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

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

Now, 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, using an instance of class `JFileChooser`, in package `javax.swing.*`. Execute the following in the interactions pane:

```
br= Lab09.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 method `getReader(p)`. Here's 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 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. Execute `jd.showOpenDialog(null)`. This 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 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, the next line of the file you selected is printed. Try it.

### Processing the lines of a file

Function `lines` in class `Lab09` 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. Here are important points:

1. The first line is read before the while loop (into 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 repeatend 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 a procedure that prompts the caller for an input file and then reads the file, printing every line that contains a '\*'. Use the statement

```
System.out.println(lin);
```

to print line `lin`. Test the function on the text file `test.txt`, which you obtained at the beginning of the lab. When you are finished, show your procedure to your TA.

### Writing files

Writing files is not much different from reading them. Look at Sec 5.10 on page 207 of the text --and the accompanying material in `ProgramLive`-- for information.