Lesson 35: Input from a Disk File

Before we look at the code necessary to input data from a disk file, let's first create a text file. We will use a text editor for this purpose. Microsoft Notepad (preferred) or Microsoft WordPad is recommended. Students sometimes have problems getting the proper extension for the file name, etc., so it is recommended that we take a brief detour to Appendix E so as to avoid problems in this area.

Create a file:

Create a text file called *MyData.in*. Store it in a folder specified by the following path (unless directed otherwise by your instructor):

C:\temp_Larry

It is assumed that your name is Larry and you have already created the folder, *temp_Larry*.

The contents of *MyData.in* should be as follows:

```
One for all and all for one.
Little House on the Prairie
11 22 33 44 55 66 77 88 << notice the spaces between the numbers
Sticks and stones
```

After the *s* in *stones* press the *Enter* key **just once**. If the programs that follow give a *NullPointerException*, then first suspect the problem of "multiple *Enter*s".

Read the file:

We finally get down to business and begin writing a class called *FileTester* that will read and display the contents of the file, *MyData.in*.

```
import java.util.*;
import java.io.*;
public class FileTester
{
        public static void main(String args[])
        {
                Scanner sf=new Scanner(new File("C:\\temp_Larry\\MyData.in"));
                … more code to come …
                sf.close( ); //We opened a file above so close it when finished.
        }
}
```

To read the file, we need to create a *Scanner* object, and this necessitates the import of *java.util.** (package name is *java.util*…see Appendix I for more on packages). *File* requires the import of java.io.*. In the above code it is the object *sf* that will be used to input data from the file.

Notice in *C:\\temp_Larry\\MyData.in* the use of the double back-slashes. Recall that \\ is the escape sequence for the back-slash character. Otherwise, if we had specified *C:\temp_Larry\MyData.in* the *t* would have been interpreted as the escape sequence for a tab …. and *M* would have been interpreted as yet another escape sequence.

Won't compile:

Unfortunately, this code won't even compile. The *File* object is capable of producing errors beyond our control. For example, suppose the file doesn't exist or is corrupted. What if we try to do successive inputs beyond the end of the file? These would all produce errors (exceptions). To correctly allow for these errors, we need to change the signature of the *main* method as follows:

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

This *throws IOException* is very mysterious right now. In a later lesson we will learn all about exceptions. For now, we simply accept by faith that *throws IOException* needs to be there in case of a file error…. If you are incurably curious and need to know, we **can** tell you briefly that for a method that is capable of throwing a **checked** exception (of which, *IOException* is a classic example) you can either handle the error in your code with *try*, *catch*, and *finally*…or you can **defer** the response to the error up the calling chain with *throws IOException*. Incidentally, the *IOException* class also requires the importing of the *java.io.** package.

At this point your *FileTester* class should compile successfully.

We will now add some more code that will actually bring in the data from the *MyData.in* file one line at a time. We will use a *while* loop to do this, and inside the loop on each iteration we will assign the lines in the text file to the *String* array, *text[ ]*. When finished with the loop, we should find that we have four array values as follows:

text[0] = "One for all and all for one."
text[1] = "Little House on the Prairie"
text[2] = "11 22 33 44 55 66 77 88"
text[3] = "Sticks and stones"

The amended class:

```
import java.io.*; // necessary File and IOException
import java.util.*; // necessary for Scanner
public class FileTester
{
	public static void main(String args[]) throws IOException
	{
		Scanner sf = new Scanner(new File("C:\\temp_Larry\\MyData.in"));

		int maxIndx = -1; //-1 so when we increment below, first index is 0
```

```
                String text[] = new String[1000]; //to be safe, declare more than we need
                while(sf.hasNext( ))
                {
                        maxIndx++;
                        text[maxIndx] = sf.nextLine( ) ;
                }
                //maxIndx is now the highest index of text[], -1 if no text lines
                sf.close( ); //We opened a file above so close it when finished.
        }
}
```

One line at a time:

A little explanation is in order. The most critical line above is *text[maxIndx] = sf.nextLine( ).* This is where we pull an entire line in from the disk file. The control part of the *while*-loop, *sf.hasNext( )*, lets us gracefully end the loop after all lines have been input.

The final version:

This is all well and good, but how do we know if it really worked? After the *sf.close( )* statement above, let's add a loop to cycle through all the appropriate *text[ ]* values and print them out. The final class is:

```
import java.io.*; // necessary for File and IOException
import java.util.*; // necessary for Scanner
public class FileTester
{
        public static void main(String args[]) throws IOException
        {
                Scanner sf = new Scanner(new File("C:\\temp_Larry\\MyData.in"));

                int maxIndx = -1; //-1 so when we increment below, first indx is 0
                String text[] = new String[1000]; //to be safe, declare plenty

                while(sf.hasNext( ))
                {
                        maxIndx++;
                        text[maxIndx] = sf.nextLine( ) ;
                }
                //maxIndx is now the highest index of text[], -1 if no text lines
                sf.close( ); //We opened a file above so close it when finished.

                for(int j = 0; j <= maxIndx; j++)
                {
                        System.out.println( text[j] );
                }
        }
```

```
}
```

The final output:

After running this program, your printout should look like this:

```
One for all and all for one.
Little House on the Prairie
11 22 33 44 55 66 77 88
Sticks and stones
```