THIS PROCESS IS RELATIVELY SLOW VS. WRITING TO RAM!

NOT ON AP TEST

STUDENT OUTLINE

Lesson 16 – Text File I/O

INTRODUCTION: In this lesson you will learn about text file input and output using the classes `FileInput` and `FileOutput`, which are provided with the library *chn.util*. The source programs you have been writing are all stored as text files. After this lesson you will have a greater understanding of how files are created, changed, saved, and retrieved.

Sending output to disk is relatively easy. Reading text files from disk is a bit harder as we need to watch out for the *eoln* (end of line) and *eof* (end-of-file) sentinel values. After this lesson, many of the programming exercises will work with data stored in text files.

The key topics for this lesson:

A. Standard ASCII Text Files
B. Saving Text Files to Disk using `FileOutput`
C. Reading Text Files from Disk using `FileInput`
D. Streams and Filters

VOCABULARY:

TEXT FILE	SENTINEL VALUE
EOF	EOLN
EXTERNAL FILE	INTERNAL FILE
FILTER	

DISCUSSION:

A. Standard ASCII Text Files

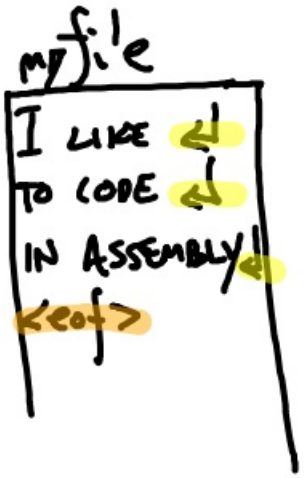
THIS IS THE ONE FOR US!

1. A text file is a sequential access file that stores characters. A sequential file is one that must be read from the beginning of the file, and then element by element. You must read the file from the beginning. You cannot jump to any location in the file.

2. A random access file is one where the program can access any element in the file, given its location in the file. Random access files are available in Java, but they will not be covered in this curriculum guide.

3. An ASCII text file is stored using the ASCII codes for each keystroke. An 'a' is ASCII code 97, which is eventually stored in binary format (using ones and zeroes) on disk.

4. A text file is organized by lines, just as a printed page would be. Each line ends with a special character (or characters), the *eoln* (end-of-line) marker. Different operating systems (Windows, MacOS, Unix) separate lines of text in different ways. In Java environments, the *eoln* marker corresponds to the enter key and has an ASCII value of 10 (`'\n'`), 13 (`'\r'`) or a combination of both.



5. The *eof* (end of file) marker marks the termination of a text file. The *eof* marker is a unique hexadecimal value that can be checked for using various stream processing tools.

6. A text file has as many *eoln*'s as there are lines in the file, but only one *eof* marker.

7. A text file can store numeric data, but integer and float types are converted into text format. ← NUMERALS ARE STORED AS ASCII

8. When a text file is stored on disk, it can be thought of as one long stream of ASCII codes. The following text example on screen would be stored as one long list of ASCII codes on disk.

   Apple
   IBM

   Disk text file (for Windows):

   THIS IS ACTUAL STORED AMOUNTS

65	112	112	108	101	13	10	73	66	77	"eof"
A	p	p	l	e	eoln		I	B	M	

B. Saving Text Files to Disk using the `FileOutput` Class

1. A `FileOutput` object must be declared to allow a program to work with output to a text file. For example:

   Program 16-1

```
import chn.util.*;

class SaveTextFile
{
  public static void main (String[] args)
  {
    FileOutput outFile;
    String fileName = new String("sample.txt");

    outFile = new FileOutput(fileName);
    ...
  }
}
```

   DECLARATION OF REFERENCE VARIABLE

   INSTANTIATION INITIATES CONNECT TO THE FILE

   FILE IDENTIFIER MAKES A FILE OF THAT NAME IN YOUR PROJECT FOLDER.

   `outFile` is the name you give to the output file (it can be anything you like); `fileName` is a `String`.

2. The declaration and initialization of `outFile` can be accomplished in one statement:

```
FileOutput outFile = new FileOutput("sample.txt");
```

   #1 TYPICAL USEAGE

   ALSO ALLOW USER TO CREATE FILE NAME

   Of the two methods there are advantages for each approach. The first method using 2 lines of code is more explicit and probably easier to read. The second method saves a line of code and compresses the declaration and initialization of the object into one statement. The curriculum guide will use the one line method most of the time.

POTENTIALLY DESTRUCTIVE! BE CAREFUL!

3. Note that a constructing a `FileOutput` object of the form

```
new FileOutput(fileName)
```

wipes out the contents of the file, `fileName`, if it already exists.

4. To append data to the end of an existing file, use

```
FileOutput outFile = new FileOutput(filename, "append");
```

MODE

Again, `outFile` is the name you provide to the output file; `fileName` is a String.

THESE "METHODS" "FEEL" SIMILAR TO WORKING WITH System.out... (DATA STREAMS)

5. Use `print` and `println` methods, the same as in `System.out`, to write data to a file. For example:

```
outFile.print("x = ");
outFile.println(x);
```

or

```
outFile.println("x = " + x);
```

Call `outFile.println()` to write a blank line.

ENSURES EVERYTHING GETS WRITTEN.

6. A `close()` method is provided to finalize file processing. The `close()` method should always be used when a program is done with a file. If a file is not closed, the program might end before the operating system has finished writing data to the file. The last data in the file might be lost!

PROJECT FOLDER

7. After the `FileOutput` object has been linked to an external file, it is ready to use as a stream to send output to disk. Program 16-2 is a complete program to send text output to a file in the current default directory path.

8. Remember that `FileOutput` and `FileInput` are not part of Java, but are provided with the `chn.util` library. Other books use other sources of I/O classes.

Program 16-2

```
import chn.util.*;

class HelloWorldWriter
{
  public static void main (String[] args)
  {
    FileOutput outFile;
    String fileName = new String("helloworld.txt");

    outFile = new FileOutput(fileName);

    for (int loop = 1; loop <= 5; loop++)
      outFile.println(loop + "   Hello world");
    outFile.close();
  }
}
```

ESTABLISHES THE CONNECTION WITH THE FILE

"FEELS" LIKE System.out... AND WE'RE COMFORTABLE WITH THAT

CLOSES THE CONNECTION

The effect of the above program will create a text file called *"helloworld.txt"* on the default drive. This file will look like this:

```
1   Hello world
2   Hello world
3   Hello world
4   Hello world
5   Hello world
```

Notice that the integer value of `loop` is saved on disk in text format.

C. Reading Text Files from Disk using `FileInput`

1. An input stream is an object used to transfer data from a file to memory. The syntax of declaring and initializing (opening) a `FileInput` object is identical to that of `FileOutput` objects. Once initialized, we can use all of the methods associated with the `FileInput` class to retrieve data from a text file.

2. Reading a text file assumes that the file already exists at some external location.

3. This first example program will read a text file of integer data and echo its contents to the monitor. The contents of "*data.txt*" are shown to the right of program 16-3.

Program 16-3

```
import chn.util.*;

public class ReadData
{
  public static void main(String[] args)
  {
    FileInput inFile;
    String fileName = "data.txt";
    int number;

    inFile = new FileInput(fileName);

    number = inFile.readInt();
    while(inFile.hasMoreTokens())
    {
      System.out.print(number + "   ");
      number = inFile.readInt();
    }
  }
}
```

data.txt

-2
-1
0
1
2
"eof"

Run output:

```
-2   -1   0   1   2
```

4. The sentinel value that terminates the **`while`** loop is the `eof` marker. This condition is detected using the `hasMoreTokens()` method with the statement:

```
inFile.hasMoreTokens()
```

5. The conditional loop read the 5 integers, echoed them to the screen, and terminated when the eof marker was encountered.

D. Streams and Filters

1. A *filter* is a program that solves a text file processing problem. A filter might change lowercase letters to uppercase, count the number of words in a file, or process the blank spaces in a file.

2. A filter will need to use an input stream (`FileInput`) and an output stream (`FileOutput`). This last example program illustrates the use of both types of file streams and how to pass streams to a method.

Program 16-4

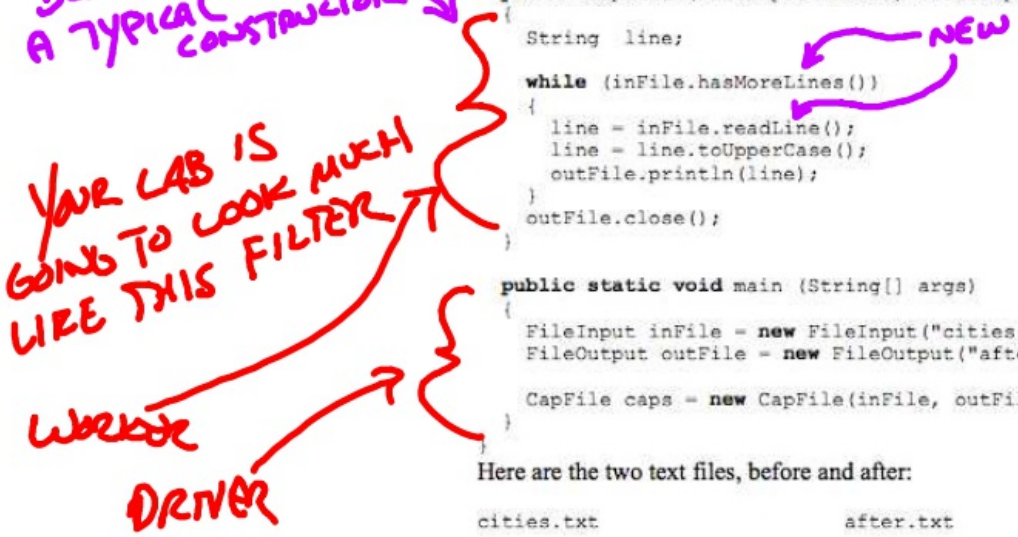
```
import chn.util.*;

public class CapFile
{
  public CapFile (FileInput inFile, FileOutput outFile)
  {
    String  line;

    while (inFile.hasMoreLines())
    {
      line = inFile.readLine();
      line = line.toUpperCase();
      outFile.println(line);
    }
    outFile.close();
  }

  public static void main (String[] args)
  {
    FileInput inFile = new FileInput("cities.txt");
    FileOutput outFile = new FileOutput("after.txt");

    CapFile caps = new CapFile(inFile, outFile);
  }
}
```



Here are the two text files, before and after:

```
cities.txt                    after.txt

San Jose                      SAN JOSE
San Francisco                 SAN FRANCISCO
```

3. Much of the lab work will involve developing process filters.

SUMMARY /REVIEW:

Text files can be used to store, manipulate, and move information from one location to another. Such files can be used to transfer information from one environment (Macintosh to Windows) and even from one application to another (word processor to spreadsheet). While there are many ways to get the job done, this lesson has attempted to show the cleanest methods to manipulate text files.

Remember that `FileOutput` and `FileInput` are not part of Java, but are provided by the *chn.util* library. Other books use other sources of I/O classes.

ASSIGNMENT: Lab Exercise L.A.16.1, *Squeeze*