IST 256
Lab
Week 9, Tuesday, March 18, 2014

1. Creating a Java application that reads and write strings from a file.

This program is a Java application that can

- read words from a file using a Scanner,
- put the words into an array of Strings,
- after the array is filled, write the words to a different file.

a. Create a **Java application** named FileReadWriteWords. As usual, the code will go into the main method of the Main class.

b. Create a Resources folder in the project. Traditionally, any data or image files used in the project are kept in a folder called Resources. Right click on the FileReadWords project name in the project pane in NetBeans and create a new **Java package** called Resources. We will put this package under the src folder in the project (the default).

c. Save a data file for the program to read.

From your browser, save the file called xanadu.txt and put it into the Resources folder for your NetBeans project.
Option 1: Use drag and drop to drag the file called xanadu.txt from the browser to your desktop, and then again use drag and drop to drag it from the desktop onto the Resources folder name in the left NetBeans pane.
Option 2: In Firefox, to save a file, right click on the link and select Save Link as … Then navigate to the folder where you keep your projects and go into the newly created folder for FileReadWords, go to src/Resources folder and save the xanadu.txt file there. (If this file doesn't show up in the NetBeans directory, you may have to close the project and re-open it.)

d. Start with the application FileWriteWords
Our program is going to be a modification of the program FileWriteWords that is on the class web page. In that program, there is one while loop that reads each word from the file, prints it to the output pane, and writes it to the file with one word per line.

At the top of the program, right after the package statement, add import statements:

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

From the web page, we click on the link called FileWriteWords and type or copy the part of the main method from the line that starts "Scanner s … " on down to the end, but not including the last two brackets. Observe the use of the Scanner functions hasNext() to see if there is another word to read, and next() to actually read the word as a String.

Run the program and observe its results. By default, the Scanner is using "whitespace" to separate words.

d. Cause an error.
In the program, change the name of the file and try to run the program. Observe the error that you get and write the name of the error here:

The instructor will demonstrate the effects of different exception methods:

```
System.out.println(e.getMessage());
System.out.println(e.toString());
e.printStackTrace();
```

e. Change the file name back to be the correct name.

f. Cause another error.
In the program, find the line with the outFile.close() method and cause it to not be executed by "commenting it out", which means to change the line to:

```
// outFile.close();
```

Rerun the program and observe the contents of the output file.

g. Change the file close statement back to the correct one.

h. Convert the application to meet the specification to save words to an array
Declare an array that can hold 100 strings and a variable to count the number put into the array:

```
// count the number of words read from the file and give the next available location
int count = 0;
// array to hold up to 100 words
String [] wordArray = new String [100];
```

In the main while loop, keep the code that reads words and prints each one, but instead of writing the word to a file, store it in the next available location in the array and increment the count.

```
// save the word to an array
wordArray[count] = word;
count++;
```

Now write a second loop, most likely a for loop, that goes over the elements of the array up to the count and not to the end, and writes those words to a file.

```
for (int i = 0; i < count; i++)
{
   outFile.write(wordArray[i]);
   outFile.newLine();
}
```

Run this program and again observe the results in the output file.

3. Writing a GUI application that reads integers from a file and processes the results

Problem Statement:

For this problem, our goal is to process a set of data consisting of the daily temperatures from one month and to find the average and the highest temperature. The program assumes that the temperatures are stored in a file called Feb05Temps.txt with one temperature per line. Any blank lines or lines with text are to be treated as comments and skipped. The user interface should have one button that reads the file and stores the temperatures in an array, one to compute the average temperature and display it, one to compute the highest temperature, and one to write the average and highest temperatures as a report to a file.

In this part, we will create a GUI application that can read one month of daily temperatures from a file.

a. Start by **creating a Java GUI application** and name it something like FileReadTemps. Then for that project, create a new jFrameForm to get the GUI window and also set the GUI to be the main class of the project.

b. Create a Resources folder in the project. Right click on the FileReadTemps project name in the project pane in NetBeans and create a new package called Resources. We will put this package under the src folder in the project.

c. Save a data file for the program to read. Copy the file called Feb05temps.txt from the folder from the web page and put it into the Resources folder for your new NetBeans project, following the steps demonstrated earlier.

d. Create the GUI:
Make a button that will read the data from the file, and a label to report the status of the file read. Also make the buttons to compute the average of the temperatures in the file and the highest temperature and labels to put the results, and make the button to write the results to a file. The form can look something like this:

|__ Read Temperatures from File Feb05temps.txt__| File Status

|__ Compute Average Temperature__| Average

|__ Find Hottest Temperature__| Hottest

|__ Write Temperature Report__| File Status

For each button, right click on Event -> Action -> actionPerformed to get a button actionPerformed method.

e. Start the GUI application:

Put the import statements at the beginning of the file before the public class statement and after the package statement, if any.

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

Then just before the first actionPerformed method, declare an array of temperatures that is of size 31. This array will be used by more than one method, so it must go at the class level, global to the methods.
This array will hold up to 31 temperatures, one per day for the longest months. But if the month is shorter (e.g. February has 28 days), not every element of the array will be filled. The variable numTemps will be used to keep the number of temperatures actually read from the file.

```
//   will hold at least 31 temperatures
int [] temperatures = new int[31];
// number of temperatures actually stored in array (between 0 and 31)
int numTemps = 0;
```

f. Write the button method that reads temperatures from a file.

This button will have code that is very similar to what we wrote in FileReadWriteWords in the part that reads from a file. Changes:

Create the BufferedReader and Scanner but not the BufferedWriter
Change the name of the input file
Change the Scanner function: instead of using next() to get a String,
use nextInt() to get an integer, something like:

```
// read the next integer in the file
temp = inputScanner.nextInt();
```

Display a file status message when done reading the file

```
// display a status message
fileStatusLabel.setText("File read done");
```

Run this program and test this button. There will be a class discussion about the control flow of this section of code, and putting comments in the temperatures file.

i. Write the button method that computes the average.
In the second button method, use the array temperatures and the number numTemps to compute the average of the array.

Computing the average will be a little different from the example that we did in the ArrayAverage program in that not all the elements of the array may be defined. You must write the loop that sums the elements of the array to go from 0 to less than numtemps. Then divide the total sum by numTemps to compute the average and put it into the label.

If you have time, you can control the **number of digits** that print after the decimal point in the average. Add a decimal formatter to this button method that prints only 3 digits after the decimal point (for example). At the top of the file, put the import statement:

```
import java.text.*;
```

In the button method, here is another way to create a formatter:

```
DecimalFormat df = new DecimalFormat("00.000");
```

Then use the function df.format to convert the average to a String with this format and display it in the label.

j. Test your program

Write the average of the temperatures in this array.

k. Add data to the file

Add another temperature to the bottom of the input file. Re-run your program and write the new number and new average here:

In Thursday's lab, we will finish this program. Wait and hand in all the code from all four button methods. This must be handed in by Tuesday, March 25.