IST 256
Lab
Week 9, Thursday, March 20, 2014

1. Extending the GUI to read temperatures from a file and report the average and high temperature.

This is the continuation of the program that you started last week called FileReadTemps with the following GUI.

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

|__ Compute Average Temperature__| Average

|__ Find Hottest Temperature__| Hottest

|__ Write Temperature Report__| File Status

b. Write the button method to compute the maximum of the array

To find the hottest temperature, we will search the array to find the maximum value of the array. Our technique will be to have an index variable that keeps track of the index of the hottest temperature that we have found so far as we go through the array. For each element, we compare it with the hottest so far, and if it is bigger, we make its index be the hottest so far. At the end, we will have found the maximum. (Refer to slides on arrays.)

Test your program.

d. Write a temperature report to an output file for the fourth button.

Now we are going to write the average and hottest temperatures in a report to an output file.

Note that we don't need to add another import statement because we already added import java.io.*;

In order to use the average and hottest temperature values in our report, the fourth button needs to be able to get the values of those variables. Therefore, we must move the declarations of the variable "average" and the variable "hottestTemp" to the class level, so that they are global to the buttons.

Also move the line with the creation of a decimal formatter from inside the Average button to the class (global) level.

In order to write the fourth button method, you may refer to the program for FileReadWriteWords.

1. Write a try catch block
2. inside the try block, create an output Stream to a file.

3. Write new code, different from FileReadWriteWords, for the output report.

    a. first write a title line that is just text. Here is an example:
    // write a title line in the file with blank lines
    outStream.write("February 2005 Temperature Report");

    b. next write two newlines

    c. write the average to the report in the file, using outStream.write, and writing a label and the average value in the same way that you would write it to System.out.println.

    d. write the hottest temperature to the report in the file:

    e. the last thing inside the try block is to close the output file:

    outStream.close();

Test your program and view the results of the file.

2. Writing a program that reads data from a CSV file

In this program, we will write a Java GUI application that can read restaurant review data from a file and display the data on the form.

a. Start by **creating a Java GUI application** and name it something like FileReview. 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 FileReview 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).

b. Prepare a data file for the program to read
Right click on the project name again and create a new empty file, using New -> Other -> Empty File. In the chooser window, create a file name, something like reviews.txt, and select the folder to be src -> Resources. (Note that if you prepare data files outside of NetBeans, you can use any other text editor, like NotePad, that doesn't add extra formatting. Do not use Microsoft Word!)

On each line of the file, type the name of a person to be the reviewer, just the first name is fine, followed by a comma, followed by the name of the restaurant they reviewed and the number of stars that they give the restaurant, a number from 1 to 5. Type in 5 or 6 lines of data. Here is an example, but feel free to use your own names and restaurants:

Anusha,Funk N Waffles,5
Jerrod,Varsity Pizza,4
Megan,Dinosaur BBQ,5
Tanner,Varsity Pizza,5
William,Dinosaur BBQ,3

We are going to read this file as a comma-separated file with the Scanner class, so do NOT type any spaces before or after any of the commas, especially the numbers.

Whenever you click Run to run the program, NetBeans will save the file before it runs the program.

c. Write the GUI application that reads restaurant reviews from a file.
Make a Label and a Button on the form for the FileReview project that looks something like:

|__Read Restaurant Review File and List Reviews__|

Reviews:

Leave lots of room for the Reviews label, so that you can put several lines of text there from the file. Make an Event -> action -> actionPerformed method for the button.

d. Write the program that will read the data file and display results.

At the top of the program, before the public class statement, put the import statements:

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

Start writing the button method.

And declare a String to display the results in the label.

```
// add <html> tag for multiline form label
String message = "<html>Restaurant Reviews: <p>";
```

Write a try/catch statement that does the following:
First it creates the input Stream and the Scanner. For the Scanner named "sc", set up the comma separated delimiters by calling the useDelimiter method:

```
BufferedReader in
        = new BufferedReader(new FileReader("src/Resources/reviews.txt"));
Scanner sc = new Scanner(in);
// use comma as a delimiter
sc.useDelimiter(",|(\\n|\\r)+");
```

Now write the while loop to read each of the three items from a line of the file. I have started the while loop for you with the code to read the first item from the file. You must add the code to read the other two items on each line. Notice that you use

- hasNext() and next() when the next item in the Stream is a String,
- hasNextInt() and nextInt() when the next item in the Stream is an integer,
- hasNextDouble() and nextDouble() when the next item in the Stream is a double.

```
// each loop iteration will read one line of the file
//    each new non-empty line will begin with a string
while(sc.hasNext())
{
   // read the person name as a String (into the variable person)
   if (sc.hasNext())
   {     person = sc.next();
         // System.out.println("item 1 " + person);
   }
   // read the restaurant name as a String (into the variable restaurant)


   // read the number of stars as an int (into the variable numStars)
```

```
        // add these items to the message string for the label display
        message = message + "<p>" + person + " reviewed " + restaurant + " with "
                + numStars + " stars";
    }
```

Notice that I added a "commented out" println commands. If your output doesn't look correct, for example, if you made a typing mistake, then you can "uncomment" the print statement to see what is being read from the file.

At the end of the try block, add the line to close the Scanner.

```
sc.close();
```

Make a catch block to catch an IOException (look at other examples for this code, e.g. from FileWriteWords or FileReadWords).

Finally, **after the try/catch statement**, set the message string as the text of the label that will display it.

```
// display the message in the label
jLabel1.setText(message);
```

Test your program.

From today's lab, hand in the button methods that you wrote for the FileTemps project and the output file that was written. Also hand in the button method that you wrote for the FileReviews program. This must be handed in by Tuesday, March 25.