IST 256
Lab Week 9, Monday, October 24, 2011

1. Writing a program that reads and writes words to a file
In this program, we will write a Java application that can read words from the file "xanadu.txt" and write them to a file "xanaduwords.txt" where each word appears on a line by itself.

a. Start by **creating a Java application** and name it something like FileWriteWords. As usual, NetBeans will bring up a program source code window where there is a class called Main and a main method where we will put our code to read a file.

b. Create a Resources folder in the project. Right click on the FileWriteWords 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. Save a data file for the program to read. From your browser, save the file called xanadu.txt from last week and put it into the folder for your NetBeans project. (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 src/Resources folder for FileWriteWords and save the xanadu.txt file there. (Or copy the file from your NetBeans project FileReadWords from last week.)

c. Write the application that reads and writes words.
At the top of the program, right after the package statement, add import statements:

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

From today's web page click on the link called FileWriteWords. 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.

Paste this program into your NetBeans main method. Fix the tabs, spacing and matching brackets. Observe the use of the output file to write out the words.
Run the program and observe its results in the output pane of the NetBeans Window.

d. Look at the output file
Observe that the output file appeared under the Resources folder in the NetBeans project FileWriteWords. Open the file xanaduwords.txt in NetBeans and observe its contents.

e. Use the write method to write more text on one line

If we want to write more text on one line, we can either catenate the Strings together with the + operator, or we can put more than one write method call before the next newline:

```
outFile.write(word + " test!");
```

e. Cause an 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.

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

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

In this program, we will write a Java GUI application that can read candy sales data from a file. In this first part, we will display the data on the form.

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

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 candy.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, just the first name is fine, followed by a comma, followed by the name of the candy bar that they sold and the number that they sold. Type in 5 or 6 lines of data. Here is an example, but feel free to use your own names:

```
Anusha,Snickers,5
Hassan,Kit Kat,2
Jamilla,Reese's Cup,3
Megan,Snickers,1
Tanner,Milky Way,4
William,Snickers,1
```

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.

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 candy sales from a file.
Make a Label and a Button on the form for the FileCandy project that looks something like:

|__Read Candy File and List Candy Sales__|

Candy Sales:

Leave lots of room for the Candy Sales 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.

First declare any variables that are needed. Here are the variables for the input file stream and the Scanner

```
 // set up input stream and scanner
Scanner sc = null;
BufferedReader in = null;
```

Add variable declarations for each item to be read from a line in the file:

```
// read file:  each line has a person name, candy name and sales number
 String person = "", candy = "";
 int numcandy = 0;
```

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

```
// add <html> tag for multiline form label
String message = "<html>Candy Sales: <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:

```
in = new BufferedReader(new FileReader("src/Resources/candy.txt"));
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 candy name as a String (into the variable candy)


   // read the number of candy sold as an int (into the variable numcandy)


   // add these items to the message string for the label display
   message = message + "<p>" + person + " sold " + numcandy + " " + candy;
}
```

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.

3. Extending the Candy program to also write data to a file

In this part of the lab, we will add code to the FileCandy application to write data to a file. We will assume that we want to write a report on which people sold a particular type of candy, and for my file, I have chosen Snickers.

All of the code will be added to the same button procedure.

a. Add a declaration for an output file variable.

```
BufferedWriter out = null;
```

b. In the same try block, add code to create the output Stream. I called my file "SnickersReport.txt", but you can make up your own output file name.

```
out = new BufferedWriter(
        new FileWriter("src/Resources/SnickersReport.txt"));
```

Before the while loop, you may also want to write title information to the file:

```
// write the title line to the file before processing lines
out.write("Snickers Report");
out.newLine(); out.newLine();
```

c. During the while loop for reading input from the file, for every person and candy that is read in, we are going to write an if test to see if the candy type is "Snickers" and we are going to write the names of those people to a file.

Using this if statement, add the file write statements that will write the person name and the new line to the file. (Look at example code.)

```
// write people who sold Snickers to the output file
   if (candy.equals("Snickers"))
   {
      // write the person name to the file, followed by a new line

   }
```

d. At the end of the try block, add a file close statement for the out Stream.

e. Run your program and **look at your output file** to see the results. If the output file doesn't show up in NetBeans under the Resources folder, close NetBeans and open it again to see the file.

Submit this lab together with the code for the button and a printout of the file that your program wrote by the beginning of class on Monday, October 31.