CIS 201 Computer Science I
Fall 2009 Lab 12
Reading Files

December 7, 2009

- Working with files (for input and output).
- Working with `String`s.
- Different kinds of loops.

This lab will take the file copying program from the last lab and extend it so that instead of just copying text from one file to another, it will modify each character and then print out the modified version of the text. That is, after reading a line, the line will be modified, character by character, into a new line. The new line will then be printed in place of the original line.

Why would you want to do something like this? Consider a text file containing something like the text for this assignment. Your program from last week could copy it but what if we wanted the copy to add terms to a search database (think **Google**). We might want to put all terms into the database in a known case (so that "mark", "Mark", and "MARK" are all stored as the same word). Converting all letters in the program to capital letters would be a good start.

Now, if you have been reading Java documentation, you might remember that there is a `String` method to convert case on a whole string at a time; you will *not* be using that method. Instead, you will build that method. Then, with just a little modification, you will convert your capitalizing program into a simple encryption/decryption tool.

Checkpoint 1 ViewFileII.java

Write a program that works just like `ViewFile.java` in the last lab *except* that it copies each input line, character by character, into another string and prints the **copy** to the screen. You should start with `EncipherFile.java` from last lab and remove all character changes.

The new program should

- Prompt the user for an input file name and read in the file name from standard input. (Consider that you must prompt the user for multiple file names soon; how can you keep from repeating yourself?)
- Open the file for input.
- If there was a problem opening the file for input, provide a useful error message to the user. Be sure to include a description of the problem and the name of the offending file. Terminate the program after printing the error message.
- If the file opened properly, copy the lines of the file to standard output; remember that `System.out` is a `PrintStream` and use a variable for output.

- You can test your program by viewing the `.java` file of the program itself.

Copying the contents of one `String` into another should not be too difficult. The following code belongs inside the loop that reads every line from the named input file. The lines (unmodified) are read into `line` and the code copies them to `modifiedLine`. Your code should then print the modified line out to the `PrintStream` (that is hooked to the screen this time).

```
String line;
// read the next line into line (inside the hasNextLine loop)...
String modifiedLine = "";
for (int characterNdx = 0;
     characterNdx < orginal.length();
     ++characterNdx) {
  char nextCh = orginal.charAt(characterNdx);
  // this is where you convert nextCh...
  modifiedLine = modifiedLine + Character.toString(nextCh);
}
// now print out modifiedLine
```

Show your work on Checkpoint 1 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 2 SwitchCase.java

Start with a copy of `ViewFileII.java`. Change the name to `SwitchCase` and make it compile.

Modify the program to copy one file to another (ask for two file names, read input line by line, and copy it character by character to a second string before writing in the output file. As each character is transferred from one string to the other, you are to change the case of the letter. All uppercase letters should be lowercase and all lowercase letters should be uppercase; everything else passes through unchanged.

You may *not* use any built-in Java `toUpper` or `toUpperCase` method (or `toLower*`).

Remember: you need to know only three things about the ASCII character encoding: lowercase letters are contiguous; uppercase letters are contiguous; digits are contiguous. How does that help you with this assignment?

You can determine whether the character being copied is an uppercase letter, a lowercase letter, or neither using routines in the `Character` class built into Java. Given that you have a lowercase letter, how do you convert it to uppercase?

Get the code for the letter (the ASCII code), add the difference between the uppercase and lowercase range of letters, and interpret the result as a character.

Alternatively, get the code for the letter, determine how far the letter is from the beginning of the lowercase alphabet, find the code for the letter the same distance from the beginning of the uppercase alphabet, and interpret that code as a character.

Huh?

Characters are stored as numbers. We want that number for our lowercase letter, say 'g'. We need to convert 'g' to 'G' so we need to get the code for 'a' and subtract it from 'g', giving us 6 (make sure you believe this is the right number). Then add 6 to the code for 'A' and interpret it as a char, or as 'G' (why *must* it be 'G'?).

Assuming, for the sake of argument, that 'a' has code 100 and 'A' has code 150 (neither number is correct), we have the following equations: just add the difference between the codes for 'A' and 'a'. So, if the code for 'a' were 100 and the code for 'A' were 150, we would have the following equation:

```
int little_a = 'a';           // little_a is 100
int big_A = 'A';              // big_A is 150

char ch = 'g';                // the character to translate
```

```
int chCode = ch;              // get the code
                              // 1111111
                              // 0000000
                              // 0123456
                              // abcdefg - g = 106

int chOffset =
  chCode - little_a;          // 106 - 100 = 6

int ucCode =
  big_A + chOffset;           // 150 + 6 = 156
                              // 1111111
                              // 5555555
                              // 0123456
                              // ABCDEFG - 156 = G
char uc = (char) ucCode;
```

Read the code again. Make sure you understand where everything came from. Now, that works for 'g'. What about 'n'? Work out, by hand, the value of `chCode`, `chOffset`, `ucCode` and `uc` (and show them to the lab monitor).

What changes to go the other way, to go from an uppercase letter to a lowercase letter? The uppercase letter's offset is found off 'A' and the lowercase letter's offset is found off 'a'. That is the only change. Write the code and trace (by hand) what happens when the current character is 'K'. Show that to the lab monitor, too.

Finally, put the translation code in to translate each letter before adding it to the translated string and writing it out to the copied file.

Show your work on Checkpoint 2 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 3 Caesar.java

Starting with a copy of `SwitchCase.java`, change the name to `Caesar` and make it compile.

Before writing the second file, each *letter* in the input file is to be enciphered using a modified Caesar cipher. In the Caesar cipher, each letter is rotated forward some number of characters (all letters move the same amount). So, if we work with the C-cipher, 'A' goes to 'C', 'B' to 'D', and so on until 'X' to 'Z', 'Y' to 'A', and 'Z' to 'B'. Every letter goes forward *modulo 26*.

You will map A to the first letter of the last name of the partner who logged on *unless* that letter is 'A' or 'M'. If both partners have an 'A' or 'M', use the first letter of a first name.

Your modified program should

- Prompt the user for an input file name and read in the file name from standard input. (Consider that you must prompt the user for multiple file names soon; how can you keep from repeating yourself?)
- Open the file for input.
- If there was a problem opening the file for input, provide a useful error message to the user. Be sure to include a description of the problem and the name of the offending file. Terminate the program after printing the error message.
- Prompt the user for an output file name and read the file name from standard input.
- Open the file for output.
- If there was a problem, give a reasonable error message and terminate the program.

- If both files opened properly, copy the lines of the input file to the output file, enciphering each letter using your given Caesar cipher. Uppercase letters should encipher to uppercase letters; lowercase letters should encipher to lowercase letters; and everything else should pass through unchanged.
- You can test your program by enciphering the `.java` file of the program itself and checking a couple of the words.

What is the Caesar cipher? It is where 'A' is converted to 'C', 'B' to 'D', and so on. It is a rotation of the alphabet. You will be writing code for a modified Caesar cipher: your code will move 'A' to the third letter in your last name (pick one author; if the letter is 'A' or your last name is too short, use 'F').

Think for a moment: What methods must you change to convert all of the letters according to your Caesar cipher?

To convert one letter: convert to an alphabet index (as in the above code), add the offset to your new first letter, and convert back to a letter. Does this make sense?

```
'd'-cipher: good dog -> jrrg grj
Plaintext:       g    o    o    d         d    o    g
Alphabet Index:  6   14   14    3    X    3   14    6
Shifted Index:   9   17   17    6         6   17    9
Ciphertext:      j    r    r    g         g    r    j
```

Encipher "Happy Holidays" using the 'd'-cipher.

Okay, what happened when you enciphered 'y'? 'y' -> 24, add three and you get 27. This is a problem. What to do? Wait, 'y', *rotated* forward 3 spots is 'b', right? And 'b' has an alphabet index of 1. And 27 *modulo* 26 is...what is it?

`27 % 26 = ?` What does it equal? Modulus gives the remainder so it would be 1. Which is the alphabet index of 'b'. So, do the arithmetic *modulo* 26.

So, you figure out the alphabet index of the letter (as in the above code), add the appropriate offset, modulo 26, and then add back the code for the appropriate letter 'a'.

Show your work on Checkpoint 3 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 4 UnCaesar.java

Starting with `Caesar.java`, you should rename it to `UnCaesar.java` and get it to compile.

How do you *decipher* a Caesar ciphered file? You modify each letter but how?

If you said "subtraction" then you're right. The only problem is that we need non-negative numbers and the `%` operator gives both negative and positive numbers. (Try printing out the value of (-1 % 26). We would like the value to be 25 (one before 0) but it comes out to -1.) That should not be a problem because we can add any multiple of 26 to our results (before the modulo operation) without changing the answer (after the modulo operation). So, subtract your shift value but add 26 to the results.

Make the program decipher according to your Caesar cipher.

Show your work on Checkpoint 4 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Log off of the lab computer you are using before leaving they lab. Anyone entering the lab has unlimited access to your files if you remain logged on. **DO NOT** turn off lab computers! They are a shared resource and there might be someone else logged in to "your" machine.