CIS 201 Fall 2008: Lab 11
Caesar Cipher

- Non-FANG Java
- Reading a file
- Writing a file

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.

At the heart of both programs is a nested loop that looks something like this (this is **pseudocode**; it will not compile but you should be able to see what is happening):

```
while (there are more lines) {
  String original = nextLine;
  String converted = "";
  while (there are characters  in original) {
    char nextCh = nextChar;
    add nextCh to end of converted;
  }
  print out converted;
}
```

Phase 1. Create a new folder and copy over one partner's copy of `CopyTextFile.java` (from the previous lab). Change the name of the copy to `CapitalizeCopy.java`. Modify the program and get it to compile.
Question 1– What is an *end-of-file-controlled loop*? Where would you find one in `CapitalizeCopy.java`?

Phase 2. Modify the EOF-controlled loop in the program to create a character-by-character copy of the line read. Look at the pseudocode above. There needs to be a loop inside of the EOF-controlled loop that converts each and every character in the `String`. If we have to do something to every character in a `String`, what should you be thinking?
You have no idea how to answer that, do you? That is, if it were an array or even an `ArrayList` you would know to think, "Use a **`for`** loop."
Since we know, before the loop starts, exactly how many times we need to do it (`String.length()` is a method that gives us...yep, the *length* of the string), you should think of using a count-controlled **`for`** loop.
Right after declaring `line` inside of the EOF-controlled loop, declare a new `String` (initialized to the empty string). Add after that a loop to extract each character from `line` using an index, `i`.

Oops, another case where you don't know *how* to do that. How to extract an element from a `String` given an index, `i`.
Question 2– How would you extract an element from an `ArrayList`, `theArrayList`, given an index `i`?
Question 3– How would you extract an element from an *array*, `theArray`, given an index `i`?
`String` is an object type (how do I know that just by reading the name? Not a question for the lab but a sure-fire question on a final exam). That means I can use the dot notation to call a method on it. So it is like the `ArrayList` answer you gave above. To print out each character in `line`, one per line, on standard output I could:

```
for (int characterNdx = 0; characterNdx < line.length(); ++characterNdx) {
  char nextCh = line.charAt(characterNdx);
  System.out.println(nextCh);
}
```

Instead of printing each character to standard output, your loop should append each character to the end of the converted string.
Finally, you should print out to the file the converted string
Compile and text your program; it should just copy files like before. The next phase will convert the case of characters.
Checkpoint 1:: Show us your code.

Phase 3. Write five methods to handle characters.
You will need to write five conversion methods. Each takes a single character as a parameter and returns a single character as its results. The reason there are five of them are one for each "class" of characters (lowercase letter, uppercase letter, digit, other) and one "dispatch" method which figures out the class of the character and calls the right one of the other four. So, let's start by writing five stub methods which just return their parameter:

```
// the dispatch method
public char convertCh(char ch) {
  return ch;
}

public char convertLC(char lower) {
  return lower;
}

public char convertUC(char upper) {
  return upper;
}

public char convertDigit(char digit) {
  return digit;
}

public char convertOther(char other) {
  return other;
}
```

Next, modify the character processing loop so that instead of just appending `nextCh`, you append the result of `convertCh` called with `nextCh`.
Fill in the dispatch method according to the following pseudocode:

```
if (ch is lowercase letter) {
  return convertLC(ch);
} else if (ch is uppercase letter) {
  return convertUC(ch);
} else if (ch is digit) {
  return convertDigit(ch);
} else {
  return convertOther(ch);
}
```

Remember: you need to know only three things about the ASCII character encoding: lowercase letters are contiguous; uppercase letters are contiguous; digits are contiguous.
After filling in the dispatch routine, compile and test your program. It should still make unchanged copies. So, what is the last step for converting lowercase letters to uppercase letters? We must get the code for the character, add to it the difference between the uppercase and lowercase alphabets, and interpret the resulting number as a char.
Huh?
Characters are stored as numbers. We want that number for our lowercase letter, say 'a'. We need to convert 'a' to 'A' so we need to get the code for 'A'. To get there we 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 lowercaseCode = 'a'; lowercaseCode = 100;
int littleaCode = 'a'; // littleaCode = 100;
// convert ASCII code to a number from 0-25; 0 for a, 1 for b, 2 for c
int alphabetIndex = lowercaseCode - littleaCode; // 100 - 100 = 0

int bigACode = 'A'; // bigACode = 150;
int resultCode = alphabetIndex + bigACode; // now code for capital letter
char result = (char)resultCode; // 150, treated as a char, or 'A'
```

Read that again. Make sure you understand where everything came from. Now, that works for 'a'. What about 'b'? Lets look at it again with a different char:

```
int lowercaseCode = 'b'; lowercaseCode = 101;
int littleaCode = 'a'; // littleaCode = 100;
// convert ASCII code to a number from 0-25; 0 for a, 1 for b, 2 for c
int alphabetIndex = lowercaseCode - littleaCode; // 101 - 100 = 1

int bigACode = 'A'; // bigACode = 150;
int resultCode = alphabetIndex + bigACode; // now code for capital letter
char result = (char)resultCode; // 151, treated as a char, or 'B'
```

Question 4– How did I know the character code for 'b' and 'B'?
Question 5– How do I know that this works for any lowercase letter?
So, you just need to change the body of `convertLC` to do the work above (without the comments because all of the numbers in them are fictional).
Compile, run, and test your program.
Checkpoint 2:: Show us your working program.

Phase 4. Copy `CapitalizeCopy.java` to `EncipherCopy.java`. Modify it so it compiles and runs.

Phase 5. Modify the code so that every letter is enciphered using a modified Caesar cipher.
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
```

Question 6– 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’.
Checkpoint 3:: Show us your code working at *enciphering* text files according to your modified Caesar cipher.

Phase 6. Copy `EncipherCopy.java` to `DecipherCopy.java`. Modify it so it compiles and runs.

Phase 7. Modify `DecipherCopy` so that instead of *enciphering* the file it *deciphers* the file. How would you undo the ciphering operation?
If you said “subtraction” and thought of the convert methods, 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 always 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.

Phase 8. Upload your Java and text files.
Make sure both authors’ names are in all of the header comments! Make sure you answered all 6 questions in the header comments of the Java files you’re uploading.
Go to Moodle; you will find an assignment in this week for the lab. Go there and upload all of the `.java` files you created today.
Also make sure that both partners have copies of the program. This program has some bearing on the current assignment so you probably want to have this code around. Comments will also be very helpful in this program for exactly that reason.