

cs171: Introduction to Programming
Programming Assignment 9

Due Date: Friday, March 31

Points: 100

For this program we will be creating puzzles known as *cryptograms*. The idea is to start with a famous quote or phrase. Then you exchange all the letters by doing a substitution. For example, change all A's to X's, then change all original B's to K's and so forth. If you have never seen cryptograms before, do a google search on 'cryptogram' and try a few puzzles yourself.

This is a moderately long program that will likely take a while. Start early and plan your time accordingly.

- Go to the class webpage and download `Cryptogram.java`. I have started the program for you with a main method. You **MUST** use this main method in your own program. Be sure to type in your name, date and description in the comment block at the top. Otherwise you should not change anything in this main method. You will only be adding methods after the main method.
- There are six methods that you need to create for this program. They are listed and described below.
 - `getQuote()`
This method prompts the user for a quote and reads it in. You will need to be sure to allow for spaces and punctuation – you might want to read more about the `Scanner` class because the usual `next()` method won't work. The other major thing you will need to do is to convert everything to upper case. The user might very well type in a mixed case message, but you will need everything to be converted to upper case. Hint: read the `String` class to see that this is very easy.
 - `getAlphabet()`
This method simply creates and allocates an array of characters. It initializes the array to hold the upper case alphabet in order ('A', 'B', ... 'Z').
 - `print()`
There are actually two different print methods. One of them accepts a string and simply prints it on one line. The second method accepts a character array and prints it on one line as well.

- `permute()`

The purpose of this method is to rearrange the order of the letters in your alphabet randomly. This algorithm is a bit tricky. You are invited to try to discover an algorithm to do this on your own. Or you can look at the end of this assignment for a description of a method you can implement.

- `scramble()`

This method accepts the permuted alphabet and the message as inputs. All upper case letters in the message are exchanged according to what's in the key. For example, if the key = ['G', 'Z', 'W', ...]. Then all A's in the message are swapped to a 'G', all B's are swapped to a 'Z' and so forth. The key tells you how to change each letter. You will not be altering the original message. Instead create a new string that contains the altered letters and return this. Important: the user's input quote may contain punctuation such as spaces, periods, quotation marks, etc. These should NOT be exchanged. You should only be exchanging the letters A-Z. Simply include all the punctuation in the scrambled output string just like it appears in the input string.

- To see how the program works, I have included a `Cryptogram.class` file on the class web page. You can download this to some directory (it will get overwritten if you put it in the same directory as your own program and then compile your program, so you might want to put it in a separate directory). This class file is the compiled version of my program. You can run it from the command line by typing: `java Cryptogram`. That way you can see how my program works if you have any doubt about the descriptions listed above.
- Be sure to comment the methods appropriately.
- Hint: remember that `chars` are really numeric types so you can do arithmetic with them. For example, try the following code:

```
char letter = 'C';  
int position = letter - 'A';
```

Here `position` will hold the value of 2. This is useful because you will need to be able to use a character to look up a position in the character array that serves as the key in order to find the scrambled version of the character.

- Here is an example of my program running. The original quote is printed first (already converted to upper case). Then the alphabet is printed. Then the permuted alphabet is printed (so you can see the transitions). Finally the quote is printed again but in scrambled form.

```

Enter a famous quote or phrase:
A ROSE BY ANY OTHER NAME WOULD SMELL AS SWEET.  -- JULIET CAPULET.
ABCDEFGHIJKLMNOPQRSTUVWXYZ
IPJLHRKANGVXYCWDOBFUMQEZTS
I BWFH PT ICT WUAHB CIYH EWMXL FYHXX IF FEHHU.  -- GMXNHU JIDMXHU.

```

Name your program `Cryptogram.java` and email me the source code before the 11:30am on Friday, March 31.

How to Permute

A *permutation* is a re-ordering of a list of things. Here we will be randomly permuting the uppercase alphabet in order to scramble the quote. You start with an array of 26 characters in order: 'A' to 'Z'. Now you need to shuffle the array so the same characters appear but in a different, random order. This can be a tricky algorithm to figure out the first time you encounter it. If you want, you may follow the pseudo-code algorithm below to implement this method.

For illustration purposes, assume we only have a five character array: (A, B, C, D, E). The same operation works on the larger full alphabet, but this method is easier to describe using the small array.

The idea is to start at the beginning with position 0 (which currently contains the A). We need to pick a random letter from anywhere to end up in the first position. This could possibly even be the A which is already there! We will generate a random number between 0 and 4 (all the positions). Then we will swap the A with whatever character is at the random position. For example, assume we randomly pick position=2. So we swap the A with the C and we now have (C, B, A, D, E). We have settled on the fact that C will come first, now we need to pick letters for the other positions.

The next step is to move to the second position (still a B). Now we select any random position from 1 to 4. We don't include the first location because we are now finished with this. Suppose we pick position=1. We essentially swap the B with itself and end up with (C, B, A, D, E).

Now we move to the third position and select a random number between 2 and 4. We swap. Continue in this fashion until you have reached the end of the array. This algorithm guarantees that each character has an equal probability of ending up in each location – we don't want an algorithm that artificially biases the shuffled order so that it isn't truly random.

Pseudocode:

Start with array a[] of size 26 with upper case letters sorted.

```
loop from position start index to position last index
  pick a random spot between current position and last index
  swap letters at position and spot
end loop
```