# ASSIGNMENT 2
# Cryptography

COMP-202, Fall 2015

Due: Wednesday, October 28th, 2015 (23:59)

You must do this assignment individually and, unless otherwise specified, you should follow all the instructions. Graders have the discretion to deduct up to 15% of the value of this assignment for deviations from the general instructions and regulations.

Before starting this assignment, you need to download the files web2.txt and `SentenceChecker.java` from mycourses. These files should be placed in the same folder as the file you will create called `Cryptography.java`

If you are working using Dr. Java, you need to open `SentenceChecker.java` at the same time. However, when you click compile and then run, you must ensure that you run your file `Cryptography.java` by leaving this window open.

| | |
|---|---|
| Part 1: | 0 points |
| Part 2, Question 1: | 15 points |
| Part 2, Question 2: | 10 points |
| Part 2, Question 3: | 20 points |
| Part 2, Question 4: | 15 points |
| Part 2, Question 5: | 10 points |
| Part 2, Question 6: | 30 points |
| | 100 points total |

## Free pass usage

If you wish to use your "free pass" (see the course webpage for more details), you are required to email the TA who is marking your assignment *before* the deadline. Please see the table below, which is referenced by the *last* name of the student, to determine your TA. Note that this is also the person whom you should see if you have questions or other complains about your marks.

| TA Name | Email Address | Student Last Name Range |
|---|---|---|
| Xiaozhong Chen | xiaozhong.chen@mail.mcgill.ca | A-BAN |
| Stephanie Laflamme | stephanie.laflamme@mail.mcgill.ca | BAO-BUDD |
| Teng Long | teng.long@mail.mcgill.ca | BUDE-CHR |
| Andrew Holliday | andrew.holliday@mail.mcgill.ca | CHS-DOW |
| Carlos Gonzalez Oliver | carlos.gonzalezoliver@mail.mcgill.ca | DOX-GAG |
| Hua Qun (Robin) Yan | huaqun.yan@mail.mcgill.ca | GAH - HARB |
| Feras Abu Talib | ta_feras@hotmail.com | HARC - JON |
| David Bourque | david.bourque@mail.mcgill.ca | JOO - KOR |
| Paul Husek | paul.husek@mail.mcgill.ca | KOS - LI, H |
| Babak Samari | babak@cim.mcgill.ca | LI, I - MA |
| Seyyed Mozafari | sh.mozafari@mail.mcgill.ca | MAC - MUR |
| Egor Katkov | egor.katkov@mail.mcgill.ca | MUS - PARE |
| Chenghui Zhou | chenghui.zhou@mail.mcgill.ca | PARF - RENT |
| Jonathan Campbell | jonathan.campbell@mail.mcgill.ca | RENU - SHIM |
| Neeth Kunnath | neeth.kunnath@mail.mcgill.ca | SHIN - TANG |
| Ayush Jain | ayush.jain@mail.mcgill.ca@mail.mcgill.ca | TANN - WANG, Q |
| Tzu-Yang (Ben) Yu | tzu-yang@mail.mcgill.ca | WANG, T - YANI |
| Mohammad Patoary | mohammad.patoary@mail.mcgill.ca | YAY- Z (end) |

## Part 1 (0 points): Warm-up

*Do* **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

**Warm-up Question 1** (0 points)
Write a method `printTypeOfChar`. This method should take as input a `char` and *print* the type of character. If the `char` is upper case, your method should print `UPPERCASE`. If the `char` is lower case, your method should print `LOWERCASE`. If the char is any other symbol, your method should print `SYMBOL`. Note that it is *not* necessary to make 52 different cases if you consider the order of the Unicode chart and the fact that you can use the < and > operators on them.

**Warm-up Question 2** (0 points)
Write a method `countUppercase`. Your method should take as input a `String` and return an `int` representing the number of upper case letters in the `String`. Now add a `main` method where you use the `Scanner` class to read a `String` from the user and call your method, outputting the returned result.

**Warm-up Question 3** (0 points)
Write a method `toUpperCase` which takes as input a `String` and returns another `String` which is the original `String` but with all lower case letters converted to upper case. Any letters that are not lower case should remain the same. **Do NOT use any methods in the Character class or the .toUpperCase() method in the String class**

For example the input `String` of “Hi!” should become “HI!”

**Warm-up Question 4** (0 points)
Write a method `reverseString` which takes as input a `String` and returns the string in reverse order. For example if the input `String` is “Comp 202 is awesome” the result should be “emosewa si 202 pmoC”

**Hint:** Use a new `String` called `reverse` and initially store into it the empty `String`. Then read the input `String` in reverse by using the method .charAt(int i) to get a specific element.

**Warm-up Question 5** (0 points)
A prime number is a positive number whose only even divisors are 1 and itself. Write a method `isPrime`

that takes as input an integer n and returns a boolean representing whether n is prime or not. Make sure to handled cases where the integer is negative. (Your method should return false in these cases.)

**Warm-up Question 6** (0 points)
Write a method firstPrimeNumbers which takes as input an int n and returns an int[] . The int[] should contain the first n prime numbers.

**Warm-up Question 7** (0 points)
Practice using the utility code we have provided by calling the method countEnglishWords() from the SentenceChecker class. To ensure things work, put all files (web2.txt, SentenceChecker.java, and your new class) into the same directory. You can then leave both SentenceChecker.java and your new class open at the same time in Dr. Java.

The method countEnglishWords takes as input a String and returns how many English words are part of it, using a list of words provided in the file web2.txt. Try calling this method to check how many English words are in the sentence The laydee hit teh man wit the bay bee.

**Warm-up Question 8** (0 points)
Practice using the Random class: Question four requires using the Random class. In this warmup question, you will generate random numbers.

Write a method that takes as input an int n and an int max and returns an int[]. The array you return should be an array of size n and filled with n random numbers between 0 and max, counting 0, but not counting max. Assume max is a positive number.

See the instructions within question 4 for help using Random class.

**Warm-up Question 9** (0 points)
Same question as before except your method should now take as input three values: int n , int min, and int max. Instead of ranging from 0 to max, you should range from min to max.

Again, you may assume both min and max are positive numbers as well as assume that min is less than max.

**Warm-up Question 10** (0 points)
$x$ is a factor of $y$ if $y$ is a multiple of $x$. Write a method calculateFactors. The method should take as input an int n and return an int[] containing all the factors of the number n

**Warm-up Question 11** (0 points)
Write a method that takes as input an array of Strings and prints all of the Strings on separate lines.

**Warm-up Question 12** (0 points)
Create a file called Counting.java, and in this file, declare a class called Counting. This class should ask the user when the computer should stop counting.

```
When should I stop counting to?
10 <---- User typed this
I am counting until 10: 1 2 3 4 5 6 7 8 9 10
```

**Warm-up Question 13** (0 points)
For this question you have to generalize the last question. The user will give you the number they want the computer to count up to and the step by which it will do so.

```
When should I stop counting to?
25 <----
Which step should I use?
3 <----
I am counting to 25 with a step of 3:
1 4 7 10 13 16 19 22 25
```

**Warm-up Question 14** (0 points)

This program should ask the user how big she wants a square to appear. Using two loops, you should be able to describe the outline of a square like the following.

```
How big do you want your square to be?
10 <---- User typed this
##########
#        #
#        #
#        #
#        #
#        #
#        #
#        #
#        #
##########
```

*N.B.* It is normal that it does not output a perfect square as the width and the length of the characters are not equal.

How would you extend your program to output a rectangle with width and length specified by the user?

# Part 2

*The questions in this part of the assignment will be graded.* All of your code in this assignment should go into a class `Cryptography`.

Beginning in this assignment, a 75% non-compilation penalty will apply. This means that if your code does not compile, you will receive a maximum of 25% for the question. If you are having trouble getting your code to compile, please contact your instructor or one of the TAs or consult each other on the discussion boards.

**For this assignment, you will be writing several methods to help you practice using arrays. As such you are not allowed to use any methods from the Java library class `Arrays`.**

## Encrypting a String With a Shift Schema

**Question 1: Encrypting a String with a Caesar shift** (15 points)

A simple way to encrypt things is by shifting all of the letters by a fixed amount $n$. This is known as a *Caesar shift.* The letters are "cycled" and all characters other than letters (e.g. numbers, punctuation, etc. remain the same).

For example, when employing a Caesar shift of 3 on the String `go Rangers!` you would get `jr Udqjhuv!`. Note that the letter 'j' comes three letters after the letter 'g' in the alphabet. The letter 'U' comes three letters after the letter 'R', and is upper-case, because the letter 'R' in this example is also upper case. We consider the alphabet of upper-case letters to be a loop (a shift of 3 applied to the letter 'Z' would give 'C'), and similarly with lower-case letters (a shift of 3 applied to the letter 'z' would give 'c').

Julius Caesar famously used this method of encoding messages. At the time, very few people were literate and those that could read simply assumed it was a foreign language! If interested, you can read more about it at `https://en.wikipedia.org/wiki/Caesar_cipher#History_and_usage`

Write a method `caesarEncrypt` which takes as input a `String originalMessage` and an `int shift` and returns the `String` created by applying a shift of size `shift` to the original message. Note that your method should work with all non-negative integers. That is, you can assume that the input will never be smaller than zero. In the case that shift is greater than 25, you should cycle over the alphabet an additional time. For example the letter *a* shifted by 30 is the same as the letter *a* shifted by 4, which would be *e*.

Two useful methods that you can use are `charAt()` and `length()` from the String class. In order to use them, write the name of the String type variable, then a dot, and then the method name followed by arguments. For example, if you have:

```
String s = ‘‘hello’’
```

then

```
s.charAt(x)
```

will return the $x^{th}$ character in s. The counting starts from 0. So for example:

```
System.out.println(s.charAt(0))
```

will print the letter h.

```
System.out.println(s.charAt(3))
```

will print the letter l.

```
System.out.println(s.charAt(5))
```

will give you a run time error because the String s does not have a character at index 5.

In addition, you can obtain the length (number of characters) of a String by using the method .length(). To do this, for example, write

```
int length = s.length()
```

In order to add a number to a character and have the result be a character, you will have to be careful with type-casting. For example:

```
char d = (char)(‘A’ + 3)
```

will will store the unicode for the letter ‘D’ in the variable d. **Clarification: The casting must be done after the addition in order for the types to work properly.**

**For this question, do not use any methods from the Java Character class. Additionally you may not use any methods from the `String` class other than those listed above (`charAt` and `length`).**

**Question 2: Decrypting a String** (10 points)

Write a method `caesarDecrypt` which takes as input a `String encoded` and an `int shift` and returns the `String` created by applying a *reverse* shift of `shift` to the original message.

`caesarEncrypt()` is the inverse of `caesarDecrypt()`. This means that for any `String s`, `caesarDecrypt(caesarEncrypt(s))` is the same as `s`

**Hint:** Can you use the method `encrypt()` that you wrote in the previous question to help for this part?

**Question 3: Cracking the code!** (20 points)

Write a method `crackCipher()` which takes as input a `String encoded` and an `int numberLetters`. It should then call the method `caesarDecrypt()` with all different shifts from $0 ... numberLetters - 1$. After getting the result from this call, it should call the method `countEnglishWords()` given in the file `SentenceChecker`. This method takes as input a String and outputs the number of English words it contains. The method provided to you will take care of things such as removing punctuation. This means, for example, that `he!!!!!!llo` will count as one English word and `hello, world :)` counts as two English words.

Your method should return a `String` representing the message decryption with the most English words. In the case of a tie, you can pick whichever one you like.

To make this work correctly, you will have to make sure that you put the file `web2.txt` as well as `SentenceChecker.java` in the same directory as your code. Both of these files can be found on MyCourses. To compile your code from the terminal, you should type `javac SentenceChecker.java Cryptography.java`. See the instructions in the warm up question if you are having trouble using this method.

## Generating a better code using permutations

Caesar's method of encryption is clearly insecure with computers. Now we will generate a more robust code, that is much more difficult to crack.

The first step is to shuffle letters in an array.

**Question 4: Shuffling** (15 points)

Write a method `shuffle` that takes an array of characters as input and returns `void`. This method should shuffle the elements in the array (remember that arrays are *reference types*, so any changes you make to the contents of the array in the method will apply in other methods as well). If you are having confusion over this, please read the class notes and examples or see a TA or instructor.

One way (and the way you will use on this assignment) to shuffle the characters of the array is by repeatedly choosing two random elements (more on this below) and swapping them with each other. For an array of length $n$, you should use $n^4$ random swaps.

To generate the two random numbers you will use Java's Random class. First, import the class, which is part of `java.util`.

Then, to create a variable of type Random (a random number generator) you need to write the following line of code. **To generate good random numbers, it is absolutely critical that the below line only be executed one time throughout your method. Otherwise, you will always generate the exact same number.**

```
Random generator = new Random(12345);
```

Note that the number 12345 is **not** a suggestion. We require that all students use the same seed in order to simplify the marking process. The *seed* determines the order of the pseudo-random numbers. You will notice that if you run your code repeatedly, you will obtain the same sequence of numbers every time. While this is sub-optimal for generating true random numbers, it is very helpful in writing code with reproducible behaviour, and in grading assignments. :)

Once you have created the `Random` object, you should use the random number generator to generate a random number between 0 (inclusive) and $n$ (exclusive) with the method **generator.nextInt(n);**.

**Question 5: Generating a Random Mapping** (10 points)

Still in the same class, write a method `generatePermutation()` which takes no input and outputs a random `char[]` . The `char[]` should be created and filled according to the following rules.

1. It should have size of exactly 26
2. It should contain each upper case English letter once and only once

**Hint:** You should use the `shuffle()` method that you wrote previously in order to do this.

**Question 6: Encrypting Using a Random Permutation** (30 points)

Write a method `permuteEncrypt()` which takes as input a `String input` and returns a String which represents an encoded `String`. You should do this by first calling the method you wrote in the previous section `generatePermutation()`. Then, for every letter, you can figure out the mapping of the $i^{th}$ letter, by figuring out the value of the $i^{th}$ element in the array.

For example, the letter $A$ is the $0^{th}$ letter in the alphabet. The character to which $A$ maps is stored in the $0^{th}$ element of the returned random `char` array.

Note that A is considered the $0^{th}$ letter, B the $1^{st}$ letter, etc until Z, which is the $25^{th}$ letter.

Note that the `generatePermutation` method you wrote returns an array with only uppercase letters in it. Your method must work on lowercase letters, but it is allowed that the lowercase letters become uppercase. Any thing that is not a letter to begin with should keep the same value.

If you encrypt this way, it is extremely difficult to guess the code. *Spiritual Growth Question (not for credit):* How many possible encryptions are there in the first method of encoding using shifts? How many possible encryptions are there in the random permutation scheme?

## What To Submit

You have to submit one zip file with all your files in it to MyCourses under Assignment 1. If you do not know how to zip files, please ask any search engine or friends. Google might be your best friend with this, and a lot of different little problems as well.

These files should all be inside your zip.

`Cryptography.java`
`confession.txt` - You should write in this file any information that you think is useful for the TA to mark the assignment. This should include things you were not sure of as well as parts of your code that you don't think it will work. Of course, like a confession, this will draw the TA's attention to the part of your code that doesn't work, but he/she will probably be more lenient than if he/she has to spend a lot of time looking for your error. It demonstrates that even though you couldn't solve the problem, you understand roughly what is going on.