REBOOT ACADEMY

Computer Training Institute
"Java Assignment 2"

Module Project – Crypto

For this project you will be writing methods of encripting and decrypting text

Part 1 - Normalize Text

The first thing we will do is normalize our input message so that it's easier to work with.

Write a method called ***normalizeText*** which does the following:

- Removes all the spaces from your text
- Remove any punctuation (. , : ; ' " ! ? ( ))
- Turn all lower-case letters into upper-case letters
- Return the result.

The call to

```
normalizeText("This is some \"really\" great. (Text)!?")
```

should return

```
"THISISSOMEREALLYGREATTEXT"
```

Part 2 - Obfuscation

We'll be turning out text into Obish. The rules for Obish are simple: you simply insert the syllable "ob" in front of every vowel sound.

So "mister ackerman" becomes "mobistober obackobermoban". This part isn't really encryption, but it does make your message a little harder for people who aren't in the know to understand.

Write a method called ***obify*** that takes a String parameter (the message to be obfuscated) and returns a string in which every vowel (A, E, I, O, U, Y) is preceded by the letters “OB” (be sure to use capital letters).

If we call obify on “THISISSOMEREALLYGREATTEXT”, it should return

```
“THOBISOBISSOBOMOBEROBEOBALLOBYGROBEOBATTOBEXT”
```

Part 3 - Unobfuscation

Write a method called ***unobify*** that takes a string in Obish and returns the string in the original language. So if you call:

```
String plainText = unobify(“OBI LOBIKOBE CHOBEOBESOBE”);
```

then plainText would store “I LIKE CHEESE”.

Part 4 - Caesar Cipher

Next we’ll be writing a Caesar Cipher. A Caesar encription "shifts" each individual character forward by a certain number or "key". Each letter in the alphabet is shifted to the letter in the alphabet that is "key" places past the original letter. With a shift value of +1, the string “ILIKEZOOS” would be rendered as “JMJLFAPPT.”

Write a method called ***caesarify*** that takes two parameters. The first argument is a string you want to encrypt, and the second is an integer that contains the shift value or "key". The function should return a string, which is the input string encrypted with the Caesar cypher using the shift value passed in its second argument. You may assume that the input string is normalized.

- Note that the alphabet “wraps around”, so with a shift value of +1 the “Z” in ZOOS became an A.
- You can also have negative shift values, which cause the alphabet to previous letters. With a -1 shift, the string “ILIKEAPPLES” would turn into “HKHJDZOOKDR.”

I will provide you with a function called ***shiftAlphabet***. This function takes one argument, an integer to specify the shift value, and returns a string, which is the uppercase alphabet shifted by the shift value. So if you call shiftAlphabet(2), you will get back the following string: “CDEFGHIJKLMNOPQRSTUVWXYZAB”

Here is the implementation for shiftAlphabet, which you can use this into your program:

```
public static String shiftAlphabet(int shift) {

    int start = 0;

    if (shift < 0) {

        start = (int) 'Z' + shift + 1;

    } else {

        start = 'A' + shift;

    }

    String result = "";

    char currChar = (char) start;

    for(; currChar <= 'Z'; ++currChar) {

        result = result + currChar;

    }

    if(result.length() < 26) {

        for(currChar = 'A'; result.length() < 26; ++currChar) {

            result = result + currChar;

        }

    }

    return result;

}
```

Part 5 - Codegroups

Traditionally, encrypted messages are broken into equal-length chunks, separated by spaces and called “code groups.”

Write a method called ***groupify*** which takes two parameters. The first parameter is the string that you want to break into groups. The second argument is the number of letters per group. The function will return a string, which consists of the input string broken into groups with the number of letters specified by the second argument. If there aren’t enough letters in the input string to fill out all the groups, you should “pad” the final group with x’s. So groupify(“HITHERE”, 2) would return “HI TH ER Ex”.

- You may assume that the input string is normalized.
- Note that we use lower-case ‘x’ here because it is not a member of the (upper-case) alphabet we’re working with. If we used upper-case ‘X’ here we would not be able to distinguish between an X that was part of the code and a padding X.

Part 6 - Putting it all together

Write a function called ***encryptString*** which takes three parameters: a string to be encrypted, an integer shift value, and a code group size. Your method should return a string which is its cyphertext equivalent. Your function should do the following:

- Call normalizeText on the input string.
- Call obify to obfuscate the normalized text.
- Call caesarify to encrypt the obfuscated text.
- Call groupify to break the cyphertext into groups of size letters.
- Return the result

Part 7 - Hacker Problem

This part is not required for course credit.

Write a method called ***ungroupify*** which takes one parameter, a string containing space-separated groups, and returns the string without any spaces. So if you call ungroupify(“THI SIS ARE ALL YGR EAT SEN TEN CEx”) you will return “THISISAREALLYGREATSENTENCE”

Now write a function called ***decryptString*** which takes three parameters: a string to be decrypted and the integer shift value used to encrypt the string, and returns a string which contains the (normalized) plaintext. You can assume the string was encrypted by a call to encryptString().

So if you were to call

```
String cyphertext = encryptString("Who will win the election?",  5,
3);

String plaintext = decryptString(cyphertext, 5);
```

… then you'll get back the normalized input string "WHOWILLWINTHEELECTION".