Program Testing Document

PROJECT #2

KAMERON DAMASKA
EECS 132

Method: alphabeticalOrder

The method takes one String parameter. If all the letters are in alphabetical order, regardless of capitalization, the method will return true. Otherwise, it will return false. Non-letter characters are ignored.

Testing for various lengths

If the String has 1 or less letters in it, the return should be true. If the String has more than 2 letters, it should check to see if the letters are in alphabetical, and return either true or false depending on the result.

```
Test 0:
> HW2.alphabeticalOrder("")
true
Test 1:
> HW2.alphabeticalOrder("a")
true
Test many:
> HW2.alphabeticalOrder("aaAaa-bbBbB-----cdefGGhij")
True
> HW2.alphabeticalOrder("aaBaa-bbBbB-----cdefGGhij")
false
```

Testing location of non-letter characters

Regardless of the location of non-letters characters, the method should return whether or not the letters in the String are in alphabetical order.

```
Test first:
> HW2.alphabeticalOrder("___aAabBbcCcxyz")
True
> HW2.alphabeticalOrder("___zaAabBbcCcxyz")
false
Test Middle:
> HW2.alphabeticalOrder("aAabB___bcCcxyz")
true
```

```
> HW2.alphabeticalOrder("zaAabB___bcCcxyz")
false
```

```
Test Last:
```

```
> HW2.alphabeticalOrder("aAabBbcCcxyz___")
true
> HW2.alphabeticalOrder("zaAabBbcCcxyz___")
false
```

Method: caesarCipher

The method takes a String parameter and an int parameter. A “Caeser Cipher” is performed on the String, which shifts each letter and number by the int parameter. For example, (“A1”, 1) would return “B2”.

Testing the shift for various int values

At 0, the String returned should be identical to the String parameter. At 1, the method should return the String parameter shifted by 1 (1 to 2, m to n, etc). At z, Z, or 9, a shift of 1 will go to a, A, or 0, respectively. At n, the letters will shift shifted by n % 26 for letters and n % 10 for numbers.

```
Test 0:
```

```
> HW2.caesarCipher("A-a, 1 ", 0)
"A-a, 1 "
```

```
Test 1:
```

```
> HW2.caesarCipher("A-a, 1 ", 1)
"B-b, 2 "
> HW2.caesarCipher("Z-z, 9", 1)
"A-a, 0"
```

```
Test many:
```

```
> HW2.caesarCipher("A-a, 1 ", 25)
"Z-z, 6 " //Only 10 numbers, so return 25 % 10 = 5
> HW2.caesarCipher("A-a, 1 ", 26)
"A-a, 7 "
```

Testing the shift for various characters

Each character, regardless of where it falls in alphabetical or numerical order, should be shifted to a character n places higher than it. Exceptions for z, Z, and 9, which loop back to the lowest values

```
Test first:
> HW2.caesarCipher("A-a, 1 ", 1)
"B-b, 2 "
Test middle:
> HW2.caesarCipher("L-l, 5 ", 1)
"M-m, 6 "
Test last:
> HW2.caesarCipher("Z-z, 9 ", 1)
"A-a, 0 "
```

Testing for repeating letters, numbers, and non-letters

Regardless of the length of the String or position of characters, the caesar cipher should be performed on all letters and numbers in the String.

```
Test First:
> HW2.caesarCipher("___aaA 111", 1)
"___bbB 222"
Test Middle:
> HW2.caesarCipher("mmM___ 555", 1)
"nnN___ 666"
Test Last:
> HW2.caesarCipher("zzZ 999___", 1)
"aaA 000___"
```

Method: repeatChars

The method takes a String parameter and an int parameter, and returns a String. The returned String has every character in the String parameter repeated a number of times equal to the int parameter. The int parameter should not be negative.

Testing the number of times each character is repeated

At 0, each character will be repeated 0 times. This should return an empty String. At 1, each character is repeated once and the returned String should be identical to the String parameter. At n, each character should be repeated n number of times

```
Test 0:
> HW2.repeatChars("Good day!", 0)
""
Test 1:
> HW2.repeatChars("Good day!", 1)
"Good day!"
Test Many:
> HW2.repeatChars("Good day!", 3)
"GGGooooooddd   dddaaayyy!!!"
```

Testing repeating characters in the String parameter at various locations

The method should repeat each character in the String, regardless of whether or not it's already a repeated character. "ooo" repeated twice should return "oooooo".

```
Test First:
> HW2.repeatChars("ooo Good Day!", 2)
"oooooo  GGoooodd  DDaayy!!"
Test Middle:
> HW2.repeatChars("Good ooo Day!", 2)
"GGoooodd  oooooo  DDaayy!!"
Test Last:
> HW2.repeatChars("Good Day! ooo", 2)
"GGoooodd  DDaayy!!  oooooo"
```

Method: wordLengths

Takes a String parameter and returns a String. The String that is returned has contains the length of each word as a number while retaining all non-letter characters. A word is a contiguous sequence of letters (“Dont” is 4 letters long, whereas “don’t” is 2 words).

Testing number of letters in each word

If there are no letters in the String, all that should be returned is the non-letter characters. If the String has no contiguous sequences of letters, at most “1” should be returned, along with non-letter characters. If the String has contiguous sequences of letters, then the String should return the lengths of those sequences, along with non-letter characters.

```
Test 0:
> HW2.wordLengths(" - , ... .")
" - , ... ."
Test 1:
> HW2.wordLengths(" a-b z, s... s.")
" 1-1 1, 1... 1."
Test Many:
> HW2.wordLengths(" In-between this, and... that.")
" 2-7 4, 3... 4."
```

Testing non-letter characters in various locations

The location of non-letter characters should have no effect on the functionality of the method, regardless if they appear at the start, in the middle, or at the end of the String.

```
Sample:
> HW2.wordLengths("Supercalifragilisticexpialidocious --
- Even though the sound of it is something quite
atrocious!")
"34 -- 4 6 3 5 2 2 2 9 5 9!"
Test First:
>HW2.wordLengths("______Supercalifragilisticexpialidoci
ous -- Even though the sound of it is something quite
atrocious!")
"______34 -- 4 6 3 5 2 2 2 9 5 9!"
```

```
Test Middle:
> HW2.wordLengths("Supercalifragilisticexpialidocious --
- Even though_____ the sound of it is something quite
atrocious!")
"34 -- 4 6_____ 3 5 2 2 2 9 5 9!"

Test Last:
> HW2.wordLengths("Supercalifragilisticexpialidocious --
- Even though the sound of it is something quite
atrocious!_____")
"34 -- 4 6 3 5 2 2 2 9 5 9!_____"
```

Method: repeatWords

The method takes a String parameter and an int parameter and returns a String. The String that is returned has each word in the String parameter repeated a number of times equal to the int parameter. A space is placed between each repeated word. All other characters are left unchanged.

Tests the number of times each word is repeated

At 0, each word should be repeated 0 times, which means there should be no words in the returned String. At 1, each word should appear once, which means the returned String should be identical to the String parameter. At n, each word should be repeated n times, with a space between each repeated word.

```
Test 0:

> HW2.repeatWords("'How are you?', I asked.", 0)
"'  ?',  ."

Test 1:

> HW2.repeatWords("'How are you?', I asked.", 1)
"'How are you?', I asked."

Test Many:

> HW2.repeatWords("'How are you?', I asked.", 2)
"'How How are are you you?', I I asked asked."

> HW2.repeatWords("'How are you?', I asked.", 5)
"'How How How How How are are are are are you you you
you you?', I I I I I asked asked asked asked asked."
```

Test the location of non-letter characters

The location of non-letter characters should have no effect on the functionality of the method, regardless if they appear at the start, in the middle, or at the end of the String.

```
Test First:
> HW2.repeatWords("______How are you?, I asked", 1)
"______How are you?, I asked"
Test Middle:
> HW2.repeatWords("How are you?______, I asked", 1)
"How are you?______, I asked"
Test Last:
> HW2.repeatWords("How are you?, I asked______", 1)
"How are you?, I asked______"
```

Method: remap

The method takes two String parameters and returns a String. The output is the first String, except with remapped characters. All ‘a’s and ‘A’s in the first String parameter are replaced by the first character in the second String parameter, all ‘b’s and ‘B’s are replaced by the second character, and so forth. All non-letter character in the first String parameter are left unchanged. All characters beyond the 26th character in the second String parameter are ignored.

Testing the length of the first String parameter

If the String parameter has a length of 0, the returned String should be length 0 aswell. For lengths greater than 0, the method should remap the first String parameter as described above

```
Test 0:
> HW2.remap("", "a")
""
Test 1:
> HW2.remap("a", "b")
"b"
```

```
> HW2.remap("b", "x")
"b"
```

```
Test Many:
```

```
> HW2.remap("The quick brown fox jumps over the lazy
dog.", "XxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXx")
"xxX XXXXX xxXXx xXx xXXxX XxXx xxX xXxX xXX."
```

Testing the length of the second String parameter

If the second String parameter a length of zero, the returned String should be indentical to the first String parameter. If the second String parameter has a length of n, all letter up to the nth letter should be remapped onto the returned String. If the second String parameter exceeds a length of 26, all character beyond the 26th are ignored.

```
Test 0:
```

```
> HW2.remap("The quick brown fox jumps over the lazy
dog.", "")
"The quick brown fox jumps over the lazy dog."
```

```
Test 1:
```

```
> HW2.remap("The quick brown fox jumps over the lazy
dog.", "1")
"The quick brown fox jumps over the l1zy dog."
```

```
Test Many (27):
```

```
> HW2.remap("The quick brown fox jumps over the lazy
dog.", "abcdefghijklmnopqrstuvwxyz1")
"the quick brown fox jumps over the lazy dog."
```

```
> HW2.remap("The quick brown fox jumps over the lazy
dog.", "zyxwvutsrqponmlkjihgfedcba1")
"gsv jfrxp yildm ulc qfnkh levi gsv ozab wlt."
```

Testing the location of non-letter characters in first String

The location of non-letter characters should have no effect on the functionality of the method, regardless if they appear at the start, in the middle, or at the end of the String.

```
Test First:
```

```
> HW2.remap("______The quick brown fox jumps over the
lazy dog.", "abcdefghijklmnopqrstuvwxyz")
"______the quick brown fox jumps over the lazy dog."
```

```
Test Middle
> HW2.remap("The quick brown fox______ jumps over the
lazy dog.", "abcdefghijklmnopqrstuvwxyz")
"the quick brown fox______ jumps over the lazy dog."
Test Last:
> HW2.remap("The quick brown fox jumps over the lazy
dog.______", "abcdefghijklmnopqrstuvwxyz")
"the quick brown fox jumps over the lazy dog.______"
```

Testing the method with various alphabetical characters

If the length of the second String parameter is at least 26, every alphabetical should be remapped.

```
Test First
> HW2.remap("a", "zyxwvutsrqponmlkjihgfedcba")
"z"
Test Middle:
> HW2.remap("m", "zyxwvutsrqponmlkjihgfedcba")
"n"
> HW2.remap("n", "zyxwvutsrqponmlkjihgfedcba")
"m"
Test Last:
> HW2.remap("z", "zyxwvutsrqponmlkjihgfedcba")
"a"
```

Method: cryptoquip

The method takes a String parameter and returns a String. Every letter in the first String is remapped to another, random letter. All same letters must be mapped to the same letter (e.g. All ‘a’s get mapped to ‘g’) and there must be one-to-one correspondence: two different letters cannot be mapped to the same thing and all 26 letters must be mapped to something.

Testing the length of the String parameter.

If the String parameter’s length is 0, the returned String should have a length of 0. At length 1, the program acts similarly to a random character generator. At n, every letter should be mapped to a new letter.

```
Test 0:
```

> HW2.cryptoquip("")
""

Test 1:

> HW2.cryptoquip("a")
"n"

> HW2.cryptoquip("a")
"s"

> HW2.cryptoquip("a")
"m"

> HW2.cryptoquip("a")
"p"

Test Many:

> HW2.cryptoquip("The quick brown fox jumps over the lazy dog")
"mfk byroa xencg vnu pytzh nlke mfk widj qns"

> HW2.cryptoquip("The quick brown fox jumps over the lazy dog")
"qbs rghyd wxlez klj tgacp lvsx qbs noum ilf"

> HW2.cryptoquip("The quick brown fox jumps over the lazy dog")
"hyt gxfql uriab jim zxsvn iptr hyt dkew oic"

Testing the location of non-letter characters:

The location of non-letter characters should have no effect on the functionality of the method, regardless if they appear at the start, in the middle, or at the end of the String.

```
Test first:
> HW2.cryptoquip("___aA")
"___mm"
```

```
Test Middle:
> HW2.cryptoquip("a___A")
"i___i"
Test Last:
> HW2.cryptoquip("aA___")
"qq___"
```

Testing the alphabetical letters and capital letter test:

Regardless of the location in the alphabetical, every letter should be remapped to a new, random letter. Capitalized versions of the same letter should be treated the same.

```
Test First:
> HW2.cryptoquip("aaaAAAaaa")
"eeeeeeeee"
Test Middle:
> HW2.cryptoquip("mmmMMMmmm")
"hhhhhhhhh"
Test Last:
> HW2.cryptoquip("zzzZZZzz")
"gggggggg"
```