isAlphabeticalOrder

For test first I'll test a scenario in which the 2nd letter of the string is lower in value to the first character in the string. I expect it to return false. For test last I'll test a scenario in which the last letter of the string is lower in value to the second to last letter in the string. I expect it to return false. For test middle I'll test a scenario in which a letter somewhere in the middle of the string is less than the letter before it. I expect it to return false. For test 0 I'll test an empty string and I expect it to return true. For test 1 I'll test a string with one character and I expect it to return true. For test many I'll test a really long string with lots of non-letter characters in there and lots of capital and lowercase letters, and I'll have one of the letters be lower in value to the letter before it. I'll do this again but in the case that it is actually alphabetical. I expect the first one to return false and the second one to return true.

Test First
```
> HW2.isAlphabeticalOrder("baabcd")
false
```
Test Last
```
> HW2.isAlphabeticalOrder("abcdc")
false
```
Test Middle
```
> HW2.isAlphabeticalOrder("abcdcefghi")
false
```
Test 0
```
> HW2.isAlphabeticalOrder("")
true
```
Test 1
```
> HW2.isAlphabeticalOrder("a")
true
```
Test Many
```
> HW2.isAlphabeticalOrder("abcdeff1GIJkL!mNOOOpQ23rStu###VwXyyy*&zza")
false
> HW2.isAlphabeticalOrder("abcdeff1GIJkL!mNOOOpQ23rStu###VwXyyy*&zz")
true
```

removeNChar

First test first I'll test a scenario in which I want to remove the first character of the string. I expect this to return the string except without the first character of the string. For test last I'll test a scenario in which I want to remove the last character of the string. I expect this to return the string except without the last character. For test middle I'll test a scenario in which I want to remove various characters throughout the middle of the string. I expect this to return the string except without the first int number of those characters. For test 0 I'll test removing 0 amount of chars from a string. I expect this to just return the original string. For test 1 I'll test removing one instance of a char from a string. I expect this to return the string except without the first instance of that character. I'll also test removing one character from a string of length 1 and I expect this to return an empty string. For test many I'll test a scenario in which I remove more than one instance of a character from a string that's of a length more than 1. I expect this to return the string without the first int instances of that char.

<u>Test First</u>
```
> HW2.removeNChar("charsareawesome!", 1, 'c')
"harsareawesome!"
```
<u>Test Last</u>
```
> HW2.removeNChar("charsareawesome!", 1, '!')
"charsareawesome"
```
<u>Test Middle</u>
```
> HW2.removeNChar("charsareawesome!", 3, 'a')
"chrsrewesome!"
```
<u>Test 0</u>
```
> HW2.removeNChar("charsareawesome!", 0, 'a')
"charsareawesome!"
```
<u>Test 1</u>
```
> HW2.removeNChar("charsareawesome!", 1, 'a')
"chrsareawesome!"
> HW2.removeNChar("e", 1, 'e')
""
```
<u>Test Many</u>
```
> HW2.removeNChar("charsareawesome!", 2, 'e')
"charsarawsome!"
```

removeString

For test first I'll test a scenario where I remove a string from the front of another string. I expect it to return the original string without the string I removed in front. For test last I'll test a scenario where I remove a string from the end of another string. I expect it to return the original string without the string I removed at the end. For test middle I'll test a scenario where I remove a string from the middle of another string. I expect it to return the original string without the string I removed from the middle. For test 0, I'll test a scenario where I remove a string from an empty string and I expect the empty string to be returned. For test 1, I'll test a scenario where I remove a string of length one from another string and I expect it to return the string without the instances of the string of length one. I'll also test a scenario where I remove a string of length one from itself, and I expect it to return an empty string. For test many, I'll test a scenario where I remove a string of more than length one from another string and I expect it to return the original string without the instances of the other string.

Test First
```
> HW2.removeString("stringsareawesome", "strings")
"areawesome"
```
Test Last
```
> HW2.removeString("stringsareawesome", "awesome")
"stringsare"
```
Test Middle
```
> HW2.removeString("stringsareawesome", "are")
"stringsawesome"
```
Test 0
```
> HW2.removeString("", "a")
""
```
Test 1
```
> HW2.removeString("stringsareawesome", "a")
"stringsrewesome"
> HW2.removeString("a", "a")
""
```
Test Many
```
> HW2.removeString("stringsareawesome", "stringsareawesom")
"e"
```

moveAllXsRight

For test first I'll test a scenario where there are Xs in the front and other characters later in the string. I expect the two Xs to be shifted over one and the first character after them to be the first character of the returned string. For test last I'll test a scenario where there are Xs at the end and other characters in front. I expect the returned string to be the same as the inputted string. For test middle I'll test a scenario where there are Xs in the middle and I expect the Xs to shift over to the right. For test 0, I'll test a scenario where there is an empty string, and I expect the returned string to still be the empty string. For test 1, I'll test a scenario where there is a string of length one that contains X and I expect an empty string to be returned. For test many, I'll test a scenario where there is a string with many different characters and many different organizations of Xs. I expect all the Xs to shift over to the right in the returned string.

Test First
```
> HW2.moveAllXsRight('X', "XXaab")
"aXXab"
```
Test Last
```
> HW2.moveAllXsRight('X', "aabXX")
"aabXX"
```
Test Middle
```
> HW2.moveAllXsRight('X', "aXXab")
```

"aaXXb"
Test 0
```
> HW2.moveAllXsRight('X', "")
""
```
Test 1
```
> HW2.moveAllXsRight('X', "X")
"X"
```
Test Many
```
> HW2.moveAllXsRight('X', "aaaXXcabdfeXadfA143XXXXaXX")
"aaacXXabdfeaXdfA143aXXXXXX"
```

moveAllXsdown

For test first, I’ll test a scenario in which X is in the first row of the two-dimensional array. I expect the array to change so that X is shifted down as far as it can go, and the other characters are shifted accordingly. For test last, I’ll test a scenario in which X is in the last row of the two-dimensional array. I expect the array to stay the same because X can’t be shifted further down any farther. For test middle, I’ll test a scenario in which X is in a row that isn’t the first or last row. I expect the array to change so that X is shifted down as far as it can go. For test 1, I’ll test a scenario in which there is one row and one column that has the character X. I expect the array to stay the same because X is already as far down as it can go. I’ll also test a scenario in which there are multiple rows but just one column. I expect the Xs to shift down as far as they can on this column. For test zero, I’ll test an empty array; I expect this to remain the same because there is nothing to shift. I’ll also test an array of empty arrays; I expect this to remain the same because there is still nothing to shift. For test many I’ll test a scenario in which there are many Xs in many different scenarios and I expect all of these Xs to shift down as far as they can go and for the other characters to shift accordingly.

Test First
```
> char[][] board = {{'a','b','c','X'},{'d','a','e','f','g'},{'i','j','i'},{'i','j','k','l'}};
> HW2.moveAllXsdown('X', board)
> board
{ { a, b, c, f }, { d, a, e, X, g }, { i, j, i }, { i, j, k, l } }
```
Test Last
```
> char[][] board = {{'a','b','c','d'},{'d','a','e','f','g'},{'i','j','i'},{'i','j','k','X'}};
> HW2.moveAllXsdown('X', board)
> board
{ { a, b, c, d }, { d, a, e, f, g }, { i, j, i }, { i, j, k, X } }
```
Test Middle
```
> char[][] board = {{'a','b','c','d'},{'d','a','e','p','g'},{'i','j','X'},{'i','j','k','p'}};
> HW2.moveAllXsdown('X', board)
> board
{ { a, b, c, d }, { d, a, e, p, g }, { i, j, k }, { i, j, X, p } }
```

Test One
```
> char[][] board = {{'X'}}
> HW2.moveAllXsdown('X', board)
> board
{ { X } }
> char[][] board = {{'a'},{'X'},{'X'},{'l'}};
> HW2.moveAllXsdown('X', board)
> board
{ { a }, { l }, { X }, { X } }
```
Test Zero
```
> char[][] board = {};
> HW2.moveAllXsdown('X', board)
> board
{  }
> char[][] board = {{},{}, {}};
> HW2.moveAllXsdown('X', board)
> board
{ {  }, {  }, {  } }
```
Test Many
```
> char[][] board = {{'a','b','c','X'},{'d','X','e','f','X'},{'X','X','i'},{'X','j','k','l'}};
> HW2.moveAllXsdown('X', board)
> board
{ { a, b, c, f }, { d, j, e, X, X }, { X, X, i }, { X, X, k, l } }
> char[][] board = {{'a','b','c','X'}};
```

moveXDownLeft

For test first, I'll test a scenario in which X is in the first row of the two-dimensional array. I expect the array to be changed so that X is shifted down and left as far as it can go and the other characters are moved accordingly. For test last, I'll test a scenario in which X is in the last row of the two-dimensional array. I don't expect the array to change at all because X is already as far down and left as it can go. For test middle, I'll test a scenario in which X is in the middle of the two-dimensional array and I expect the array to be changed so that X is shifted down as far down and left as it can go and the other characters are moved accordingly. For test one, I'll test a scenario in which there is only one row in the array, and I expect the array to not change at all because X will be as far down and left as it can be. For test zero I'll test an empty array and I expect this array to be unchanged because there is nothing to shift. For test many I'll test an array with many instances of X, and I expect only the first instance of X and those characters on its diagonal to be shifted.

Test First
```
> char[][] board = {{'a','b','c','X'},{'d'},{'e','f','g','h'},{'i','j','k'},{'l','m','n','o'}};
> HW2.moveXDownLeft('X', board)
> board
{ { a, b, c, f }, { d }, { e, i, g, h }, { X, j, k }, { l, m, n, o } }
```
<u>Test Last</u>
```
> char[][] board = {{'a','b','c','a'},{'d'},{'e','f','a','h'},{'i','h','k'},{'l','m','n','X'}};
> HW2.moveXDownLeft('X', board)
> board
{ { a, b, c, a }, { d }, { e, f, a, h }, { i, h, k }, { l, m, n, X } }
```
<u>Test Middle</u>
```
> char[][] board = {{'a','X', 'q'}};
> HW2.moveXDownLeft('X', board)
> board
{ { a, X, q } }
```
<u>Test One</u>
```
> char[][] board = {{'a','b','X', 'a'}};
> HW2.moveXDownLeft('X', board)
> board
{ { a, b, X, a } }
```
<u>Test Zero</u>
```
> char[][] board = {};
> HW2.moveXDownLeft('X', board)
> board
{  }
```
<u>Test Many</u>
```
> char[][] board = {{'a','b','c','X'},{'d'},{'e','f','X','h'},{'i','X','k'},{'l','m','n','o'}};
> HW2.moveXDownLeft('X', board)
> board
{ { a, b, c, f }, { d }, { e, i, X, h }, { X, X, k }, { l, m, n, o } }
```