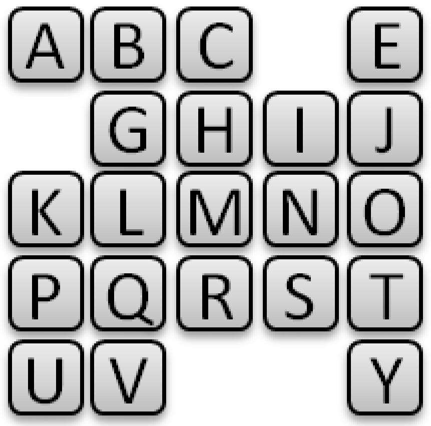
Chess Knight Move Combinations

The goal of this exercise is to see how you write code and solve problems. Beyond the requirements specified in this document, getting to the right answer is the goal. We have seen candidates solve this problem in as little as 45 minutes while most can require as much as one hour and 30 minutes. We allow one hour and 30 minutes to solve the problem as we believe it isn't too complex and the problem requirements are clear.

Pictured below is a Matrix of cells:



Your code should be written in Java and should calculate how many unique paths of 8 cells using knight moves with those properties:

- The starting cell can be any cell
- Each move **must** be a *knight move* from the current position
- Only combination of 8 moves can be counted
- No wrap around the matrix is permitted
- There can be at most 2 vowels in any combination (in our case we consider Y a vowel, therefore vowels are A,E,I,O,U and Y
- Empty cells are just that, empty, therefore should not be considered as valid moves
- Paths can be re-used as long as the constraint of total cells and uniqueness of the combination is enforced. Essentially, it is possible to have a combination of moves using the same 2 cells only.

A *knight move* is made in one of the following ways, like on a chess board:

- Move two steps horizontally and one step vertically.
- Move two steps vertically and one step horizontally.

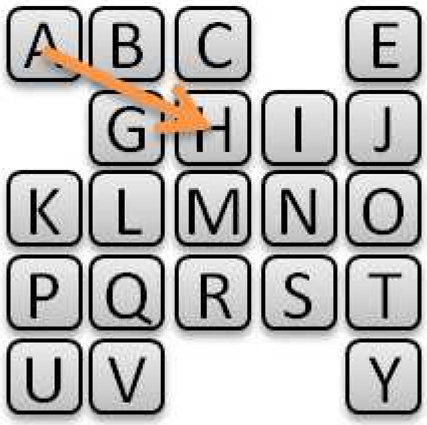
Your program should have a unit with a class called with your full name as “First+Last”, like for candidate John Doe the class and namespace will be "JohnDoe" with a method having the signature below, the return value is the count of unique paths. **To be precise all the code needs to be in one single file and not rely on any 3rd party libraries.**

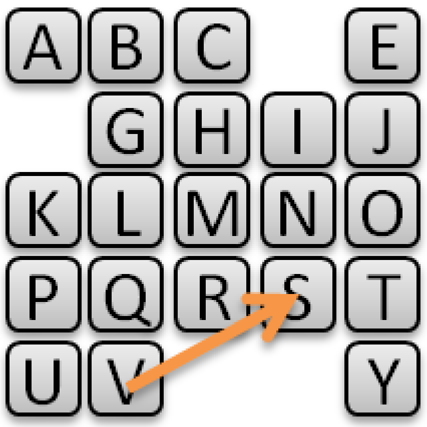
```
public static long SolveMatrix()
```

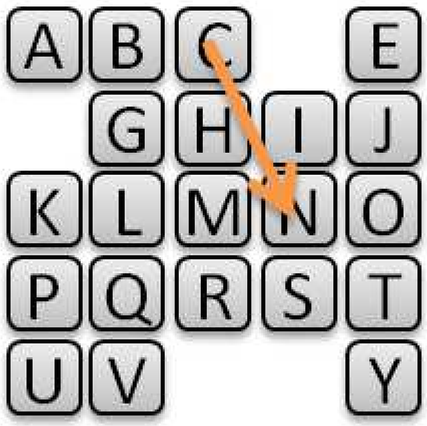
Following completion of the code, **some modifications might be required to be performed** to support different inputs but essentially performing the same functionality.

Keep in mind while coding:

- Code design should be as elegant as possible
- There are lots of way to achieve the desired result and we want to constrain you as little as possible in solving the problem.
- Code should execute as quickly as possible
- Performance considerations in the design will be discussed


A B C E
G H I J
K L M N O
P Q R S T
U V Y


A B C E
G H I J
K L M N O
P Q R S T
U V Y


A B C E
G H I J
K L M N O
P Q R S T
U V Y