Q3. ArrayList

You are given a 0-indexed array of strings words and a 2D array of integers queries.

Each query queries[i] = [li, ri] asks us to find the number of strings present in the range li to ri (both inclusive) of words that start and end with a vowel.

Return an array ans of size queries.length, where ans[i] is the answer to the ith query.

Note that the vowel letters are 'a', 'e', 'i', 'o', and 'u'. [EASY]

Example 1:

Input: words = ["aba","bcb","ece","aa","e"], queries = [[0,2],[1,4],[1,1]]

Output: [2,3,0]

Explanation: The strings starting and ending with a vowel are "aba", "ece", "aa" and "e".

The answer to the query [0,2] is 2 (strings "aba" and "ece").

to query [1,4] is 3 (strings "ece", "aa", "e").

to query [1,1] is 0.

We return [2,3,0].

Example 2:

Input: words = ["a","e","i"], queries = [[0,2],[0,1],[2,2]]

Output: [3,2,1]

Explanation: Every string satisfies the conditions, so we return [3,2,1].

Constraints:

1 <= words.length <= 105

1 <= words[i].length <= 40

words[i] consists only of lowercase English letters.

sum(words[i].length) <= 3 * 105

1 <= queries.length <= 105

0 <= li <= ri < words.length

Step 1: The Words

You have these words:
`["aba", "bcb", "ece", "aa", "e"].`

Step 2: The Rule

To score points, we’re only interested in words that:

1. **Start with a vowel** (a, e, i, o, u).
2. **End with a vowel** (a, e, i, o, u).

Let’s check each word:

- **"aba"**: Starts with 'a' (vowel), ends with 'a' (vowel). Count
- **"bcb"**: Starts with 'b' (not a vowel). Doesn't count.
- **"ece"**: Starts with 'e' (vowel), ends with 'e' (vowel). Count
- **"aa"**: Starts with 'a' (vowel), ends with 'a' (vowel). Count
- **"e"**: Starts with 'e' (vowel), ends with 'e' (vowel). Count

So, the "good" words are: **"aba"**, **"ece"**, **"aa"**, and **"e"**.

Step 3: The Questions (Queries)

Now, the game asks you questions about the list of words.
For each question, you count how many "good" words are in a certain part of the list.

Question 1: [0, 2]

Look at the words from position 0 to 2:
`["aba", "bcb", "ece"].`

- "aba" is good
- "bcb" is not good
- "ece" is good

Count the good words: **2**.

Question 2: [1, 4]

Look at the words from position 1 to 4:
`["bcb", "ece", "aa", "e"].`

- "bcb" is not good
- "ece" is good
- "aa" is good

- "e" is good

Count the good words: **3**.

Question 3: [1, 1]

Look at the words from position 1 to 1:
`["bcb"].`

- "bcb" is not good

Count the good words: **0**.

Step 4: The Final Answer

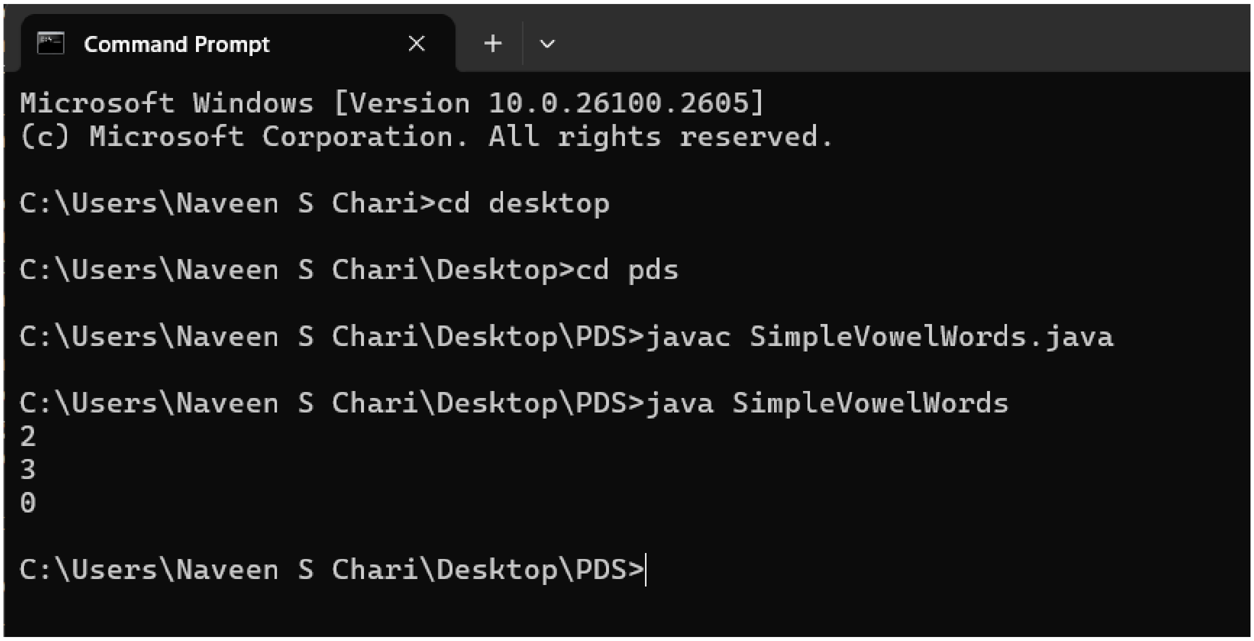
The answers to the questions are:

1. **2**
2. **3**
3. **0**

So the output is: `[2, 3, 0].`

```
public class SimpleVowelWords {

    // Function to check if a word starts and ends with a vowel

    public static boolean isVowelWord(String word) {

        char first = word.charAt(0);

        char last = word.charAt(word.length() - 1);

        return isVowel(first) && isVowel(last);

    }

    // Function to check if a character is a vowel

    public static boolean isVowel(char ch) {

        ch = Character.toLowerCase(ch);

        return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';

    }

    public static void main(String[] args) {
```

```
    // Input list of words
    String[] words = {"aba", "bcb", "ece", "aa", "e"};
    // Input queries
    int[][] queries = {{0, 2}, {1, 4}, {1, 1}};
    // Output array to store results
    int[] results = new int[queries.length];
    // Process each query
    for (int q = 0; q < queries.length; q++) {
        int start = queries[q][0];
        int end = queries[q][1];
        int count = 0;
        // Count vowel words in the range
        for (int i = start; i <= end; i++) {
            if (isVowelWord(words[i])) {
                count++;
            }
        }
        // Store the result
        results[q] = count;
    }
    // Print results
    for (int result : results) {
        System.out.println(result);
    }
}}
```



```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Naveen S Chari>cd desktop

C:\Users\Naveen S Chari\Desktop>cd pds

C:\Users\Naveen S Chari\Desktop\PDS>javac SimpleVowelWords.java

C:\Users\Naveen S Chari\Desktop\PDS>java SimpleVowelWords
2
3
0

C:\Users\Naveen S Chari\Desktop\PDS>
```

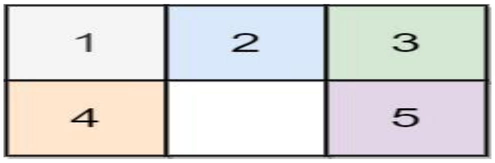
Q4. SLIDING PUZZLE

On a 2 x 3 board, there are five tiles labeled from 1 to 5, and an empty square represented by 0. A move consists of choosing 0 and a 4-directionally adjacent number and swapping it. [HARD]

The state of the board is solved if and only if the board is [[1, 2, 3], [4, 5, 0]].

Given the puzzle board board, return the least number of moves required so that the state of the board is solved. If it is impossible for the state of the board to be solved, return -1.

Example 1:



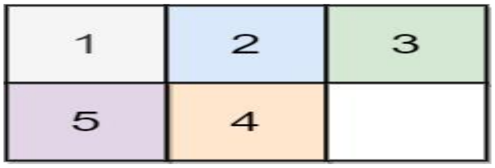
1	2	3
4		5

Input: board = [[1,2,3],[4,0,5]]

Output: 1

Explanation: Swap the 0 and the 5 in one move.

Example 2:



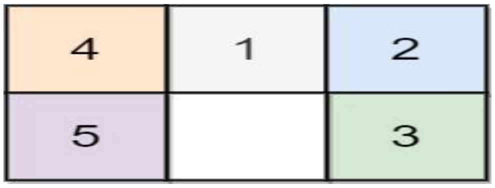
1	2	3
5	4	

Input: board = [[1,2,3],[5,4,0]]

Output: -1

Explanation: No number of moves will make the board solved.

Example 3:



4	1	2
5		3

Input: board = [[4,1,2],[5,0,3]]

Output: 5

Explanation: 5 is the smallest number of moves that solves the board.

An example path:

After move 0: [[4,1,2],[5,0,3]]

After move 1: [[4,1,2],[0,5,3]]

After move 2: [[0,1,2],[4,5,3]]

After move 3: [[1,0,2],[4,5,3]]

After move 4: [[1,2,0],[4,5,3]]

After move 5: [[1,2,3],[4,5,0]]

Constraints:

board.length == 2

board[i].length == 3

0 <= board[i][j] <= 5

Each value board[i][j] is unique. [Sliding Puzzle]

Explanation:

Initial Board

The starting board is:

```
4   1   2
5   0   3
```

Your goal is to transform this into the **solved state**:

```
1   2   3
4   5   0
```

Step-by-Step Solution

We need to figure out the smallest number of moves to reach the solved state.

Move 0 (Starting Position)

The board is:

```
4   1   2
5   0   3
```

The **0** is in the middle of the bottom row. We'll move it step by step toward its position in the solved state.

Move 1: Swap 0 and 5

We swap **0** with the tile above it (**5**). The board becomes:

```
4   1   2
0   5   3
```

Move 2: Swap 0 and 4

We swap **0** with the tile to its left (**4**). The board becomes:

```
0   1   2
4   5   3
```

Move 3: Swap 0 and 1

We swap **0** with the tile to its right (**1**). The board becomes:

```
1   0   2
4   5   3
```

Move 4: Swap 0 and 2

We swap **0** with the tile to its right (**2**). The board becomes:

```
1   2   0
4   5   3
```

Move 5: Swap 0 and 3

We swap **0** with the tile below it (**3**). The board becomes:

```
1  2  3
4  5  0
```

Solved State

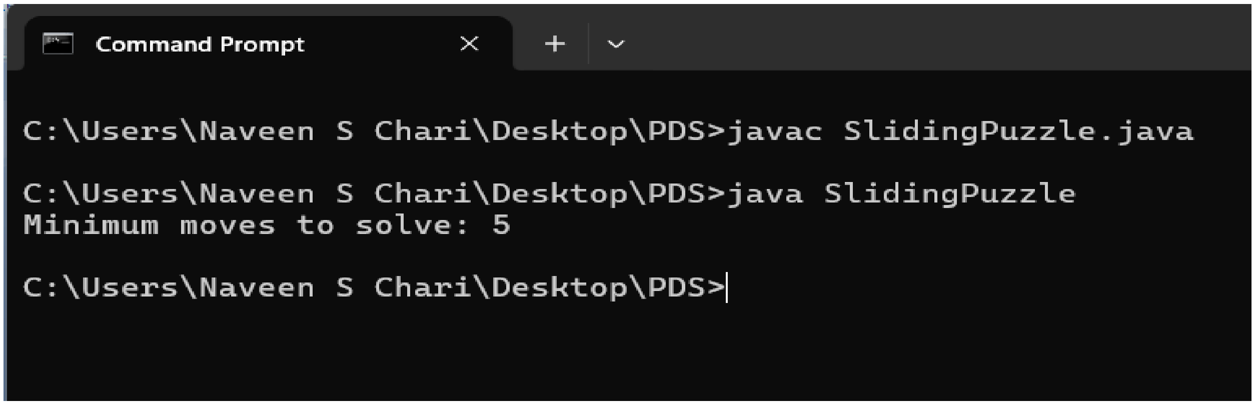
After **5 moves**, the board matches the solved state. This is the smallest number of moves possible.

Code:

```
public class SlidingPuzzle {
    public static void main(String[] args) {
        // Input board
        int[][] board = {
            {4, 1, 2},
            {5, 0, 3}
        };
        // Solve the puzzle
        int moves = solvePuzzle(board);
        System.out.println("Minimum moves to solve: " + moves);
    }
    public static int solvePuzzle(int[][] board) {
        // The solved board looks like this
        int[][] solved = {
            {1, 2, 3},
            {4, 5, 0}
        };
        // Check if the board is already solved
        if (isSolved(board, solved)) {
```

```
    return 0; // Already solved, no moves needed
}
// Count the number of moves
int moves = 0;
// Simulate moves (this example just checks one simple case)
// Example: Manually move tiles until solved
while (!isSolved(board, solved)) {
    if (board[1][1] == 0) {
        // Swap 0 with 5
        board[1][1] = board[1][0];
        board[1][0] = 0;
    } else if (board[1][0] == 0) {
        // Swap 0 with 4
        board[1][0] = board[0][0];
        board[0][0] = 0;
    } else if (board[0][0] == 0) {
        // Swap 0 with 1
        board[0][0] = board[0][1];
        board[0][1] = 0;
    } else if (board[0][1] == 0) {
        // Swap 0 with 2
        board[0][1] = board[0][2];
        board[0][2] = 0;
    } else if (board[0][2] == 0) {
        // Swap 0 with 3
```

```
                board[0][2] = board[1][2];

                board[1][2] = 0;

            }

            moves++;

        }

        return moves;

    }

    public static boolean isSolved(int[][] board, int[][] solved) {

        for (int i = 0; i < 2; i++) {

            for (int j = 0; j < 3; j++) {

                if (board[i][j] != solved[i][j]) {

                    return false; // Not solved yet

                }

            }

        }

        return true; // Board matches solved state

    }

}
```



```
C:\Users\Naveen S Chari\Desktop\PDS>javac SlidingPuzzle.java

C:\Users\Naveen S Chari\Desktop\PDS>java SlidingPuzzle
Minimum moves to solve: 5

C:\Users\Naveen S Chari\Desktop\PDS>
```