Assignment 4

Enrique Saracho Felix
100406980
CPSC 1150
14/07/2023

Exercise 1

Program ArraysExercise

File name: ArraysExercise.java

Purpose: This program can create arrays based on the number of elements and values specified by the user. It also has methods for displaying it, finding its maximum and minimum values, the indexes for those values, checking if the arrays are sorted in ascending or descending order, and more.

Packages: javax.swing.JOptionPane

Input: The program asks for integer values multiple times depending on the method being used.

Output: Depending on the method, the program will display messages containing responses with strings.

Pseudocode:

```
Algorithm ArraysExercise
START

(main)
    Set array of integers arr1 = getArray()
    printArray(arr1)

    Print findMax(arr1)
    Print findMin(arr1)

    Print findMaxIndex(arr1)
    Print findMinIndex(arr1)

    Print isSortedAscend(arr1)
    Print is SortedDescend(arr1)

    swapNeighbor(arr1)
    printArray(arr1)

    Set array of  integers arr2 = getArray()
    printArray(arr2)
    printArray(merge(arr1, arr2))
```

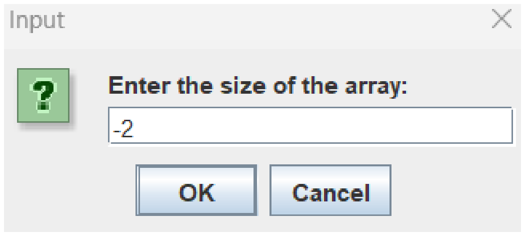
```
(getArray)
    Set integer n
    Set boolean error = false

    Do {
        If ( error = true ) {
            Print error message
        }
        Read n
        error = true
    } While ( n <= 0 )

    Set array of integers array of size n

    Set integer i = 0
    While ( i < n ) {
        Read array[ i ]
        i++
    }

    Return array

(printArray, parameter: array of integers array)
    Set integer n = array.length
    Set string elements = “”,

    Set integer i = 0
    While ( i < n ) {
        elements += array[ i ] + new line
        i++
    }

    Print elements

(findMax, parameter: array of integers array)
    Set integer max = array[0]

    Set integer i = 0
    While ( i < array.length ) {
        If ( array[ i ] > max ) {
            max = array[ i ]
        }
        i++
    }

    Return max
```

```
(findMin, parameter: array of integers array)
    Set integer min = array[0]

    Set integer i = 0
    While ( i < array.length ) {
        If ( array[ i ] < min ) {
            min = array[ i ]
        }
        i++
    }

    Return min


(findMaxIndex, parameter: array of integers array)
    Set integer n = array.length
    Set integer max = findMax(array)

    Set integer i = 0
    While ( i < n ) {
        If ( array[ i ] = max ) {
            Return i
        }
        i++
    }
    Return array[ n - 1 ]

(findMinIndex, parameter: array of integers array)
    Set integer n = array.length
    Set integer min = findMin(array)
    Set integer index = 0

    Set integer i = 0
    While ( i < n ) {
        If ( array[ i ] = min ) {
            index = i
        }
        i++
    }
    Return index

(isSortedAscend, parameter: array of integers array)
    Set integer n = array.length
    Set integer pivot = array[0]

    Set integer i = 0
    While ( i < n ) {
        If ( array[ i ] < pivot ) {
            Return false
        }
        pivot = array[ i ]
        i++
    }
    Return true

(isSortedDescend, parameter: array of integers array)
    Set integer n = array.length
    Set integer pivot = array[0]

    Set integer i = 0
    While ( i < n )  {
        If ( array[ i ] > pivot ) {
            Return false
        }
        pivot = array[ i ]
        i++
    }
    Return true

(swapNeighbor, parameter: array of integers array)
    Set integer n = array.length

    Set integer i = 0
    While ( i < n - 1 ) {
        if ( array[ i ] > array[ i + 1 ] ) {
            array[ i + 1 ] += array[ i ]
            array[ i ] = array[ i + 1 ] – array[ i ]
            array[ i + 1 ] -= array[ i ]
        }
    }

(merge, parameters: array of integers array1 and array2)
    If ( isSortedAscend(array1) = false or isSortedAscend(array2) = false) {
        Print error message
        Return empty array
    }

    Set integer n = array1.length
    Set integer m = array2.length
    Set integer l = n + m
    Set array of integers merged of size l

    Set integer i = 0
    While ( i < l ) {
```

```
                If ( j < n and k < m) {
                        If ( array1[ j ] < array2[ k ] ) {
                                merged[ i ] = array1[ j ]
                                j++
                        } Else {
                                merged[ i ] = array2[ k ]
                                k++
                        }
                } Else {
                        if ( j = n and k < m ) {
                                merged[ i ] = array2[ k ]
                                k++
                        }
                        if ( k = m and j < n ) {
                                merged[ i ] = array1[ j ]
                                j++
                        }
                }
                i++
        }
        Return merged

END ArraysExercise
```

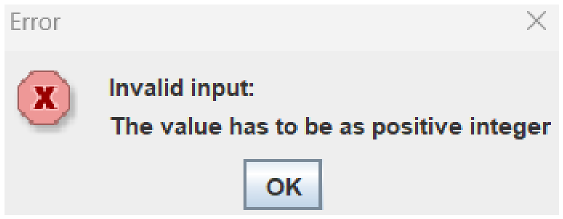
Test run(s):

Testing getArray():


Input

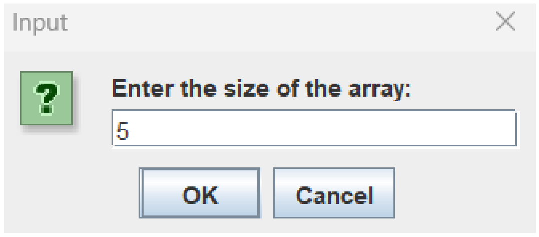
Enter the size of the array:

-2


Error

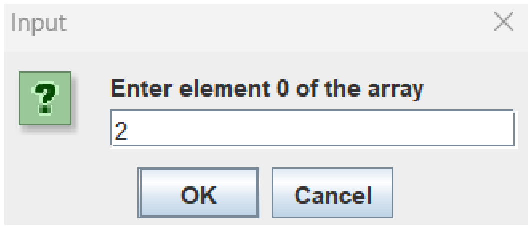
Invalid input:

The value has to be as positive integer


Input

Enter the size of the array:

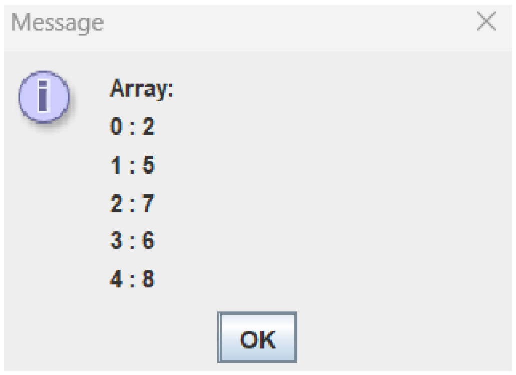
5


Input

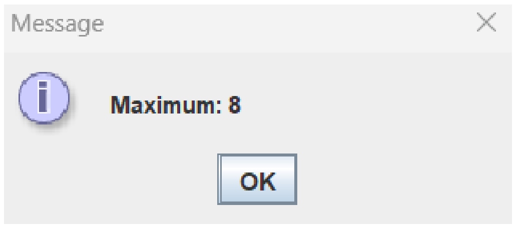
Enter element 0 of the array

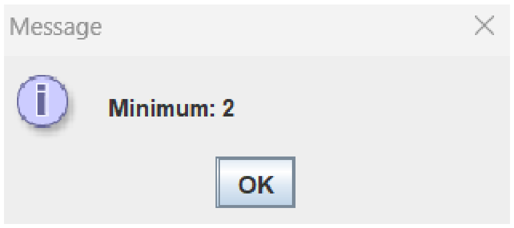
2

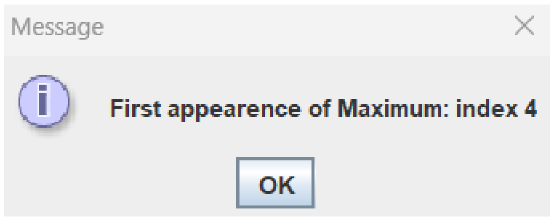
Testing printArray():

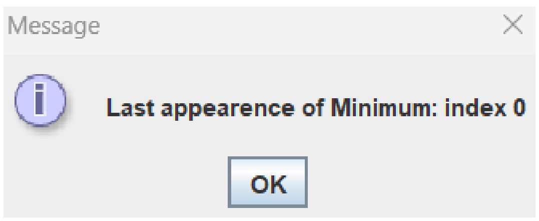

```
Array:
0 : 2
1 : 5
2 : 7
3 : 6
4 : 8
```

Testing findMax(), findMin(), findMaxIndex(), and findMinIndex():

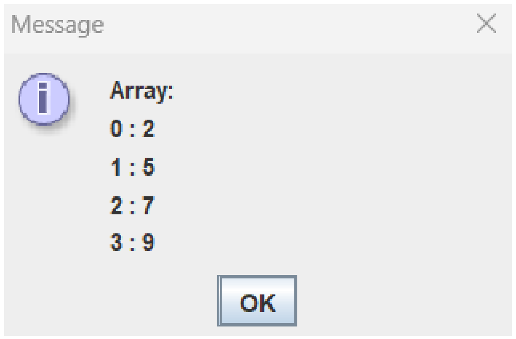

```
Maximum: 8
```

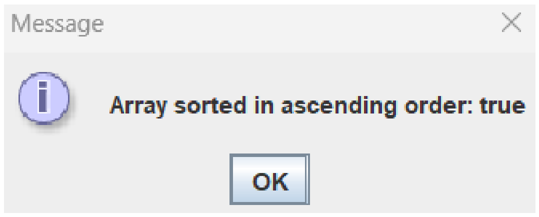

```
Minimum: 2
```

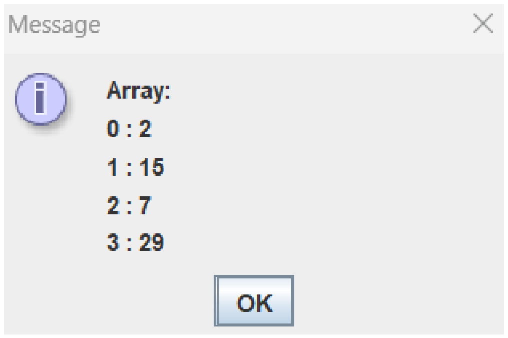

```
First appearence of Maximum: index 4
```

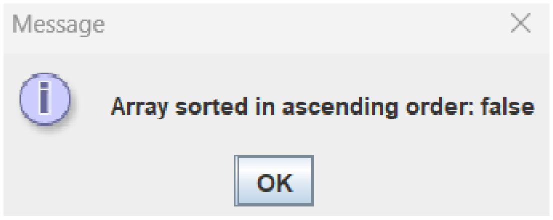

```
Last appearence of Minimum: index 0
```

Testing isSortedAscend():

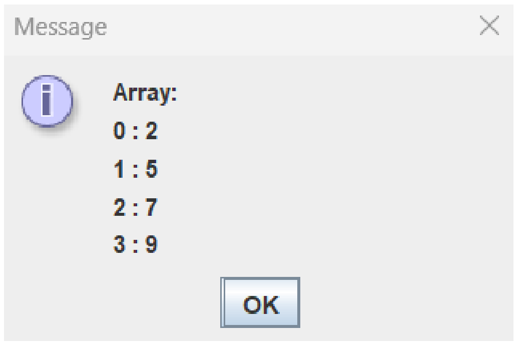

```
Array:
0 : 2
1 : 5
2 : 7
3 : 9
```


```
Array sorted in ascending order: true
```

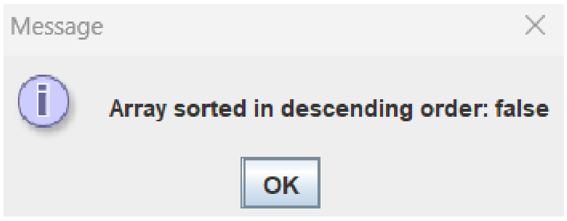

```
Array:
0 : 2
1 : 15
2 : 7
3 : 29
```


```
Array sorted in ascending order: false
```

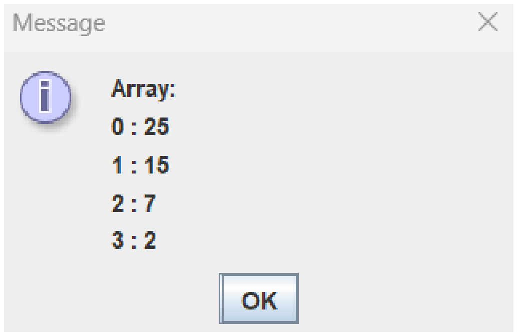
Testing isSortedDescend():



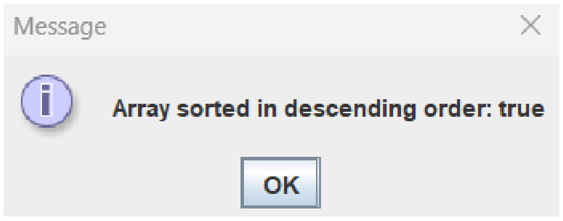
```
Array:
0 : 2
1 : 5
2 : 7
3 : 9
```



```
Array sorted in descending order: false
```

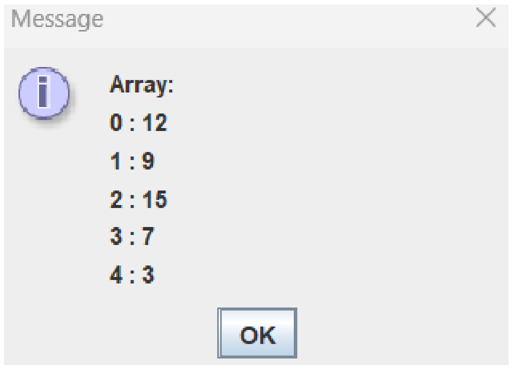


```
Array:
0 : 25
1 : 15
2 : 7
3 : 2
```

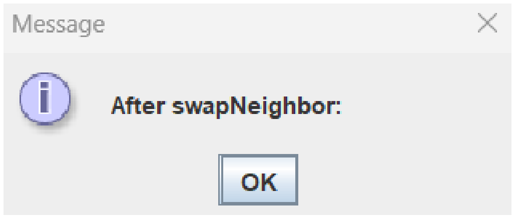


```
Array sorted in descending order: true
```

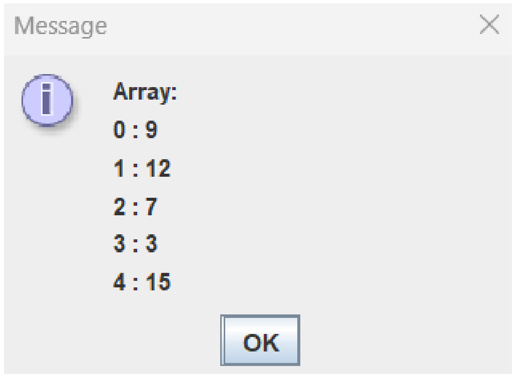
Testing swapNeighbor():



```
Array:
0 : 12
1 : 9
2 : 15
3 : 7
4 : 3
```

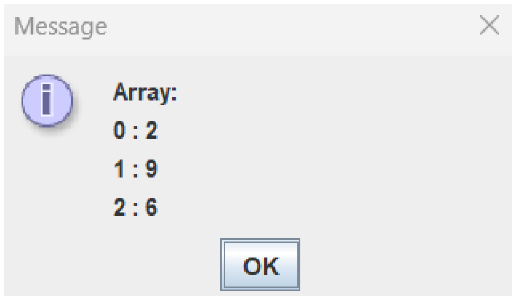


```
After swapNeighbor:
```

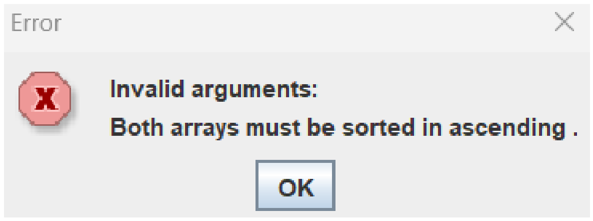


```
Array:
0 : 9
1 : 12
2 : 7
3 : 3
4 : 15
```

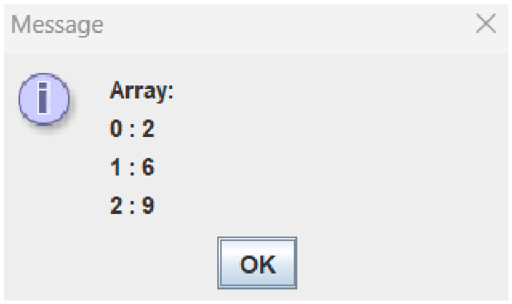
Testing merge():



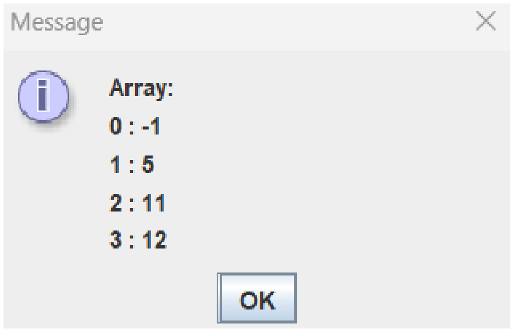
```
Array:
0 : 2
1 : 9
2 : 6
```



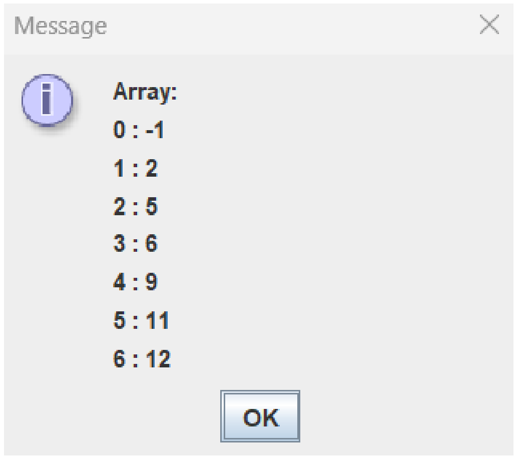
```
Invalid arguments:
Both arrays must be sorted in ascending .
```



```
Array:
0 : 2
1 : 6
2 : 9
```



```
Array:
0 : -1
1 : 5
2 : 11
3 : 12
```



```
Array:
0 : -1
1 : 2
2 : 5
3 : 6
4 : 9
5 : 11
6 : 12
```

Exercise 2

Program SecretPhrase

File name: SecretPhrase.java

Purpose: To allow the user to play a game in which the user tries to guess a random phrase in the least amount of tries possible, displaying the phrase's letters replaced by asterisks as a hint.

Packages: import javax.swing.JOptionPane;

Input: A character, multiple times, until the user enters all the characters in the secret phrase.

Output: A string, containing the phrase transformed into uppercase, with the unguessed letters replaced by asterisks. And another string at the end with the score of the game and the original phrase.

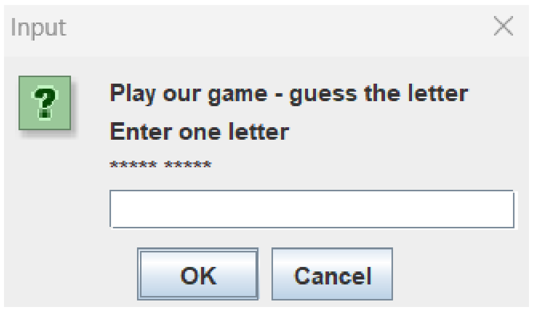
Pseudocode:

```
Algorithm SecretPhrase
START

        string[ ] phrases = {(10 phrases)}

(main)
        string phrase = phrases[random index]
        character[ ] guesses = array of characters
        integer guess = 0
        integer tries = 0

        while ( replaceLetters(phrase, guesses) != phrase ) {
                guesses[guess] = getInput(replaceLetters(phrase, guesses))
                guess += 1
                tries += 1
        }

        float score = phrase.length(without spaces) / tries

        print("Congrats!" + phrase + score)

(replaceLetters, parameters: string phrase, character[ ] guesses)
        string replacedPhrase = ""
        for ( i from 0 to phrase.length ) {
                if ( phrase[ i ] == " " )
                        replacedPhrase += " "
                else if ( findCharacter(phrase[ i ], guesses) )
                        replacedPhrase += phrase[ i ]
                else
                        replacedPhrase += "*"
        }
        return replacedPhrase

(getInput, parameter: string phrase)
        character guess
        print phrase
        read guess
        uppercase(guess)
        return guess


(findCharacter, parameters: character letter, character[ ] guesses)
```

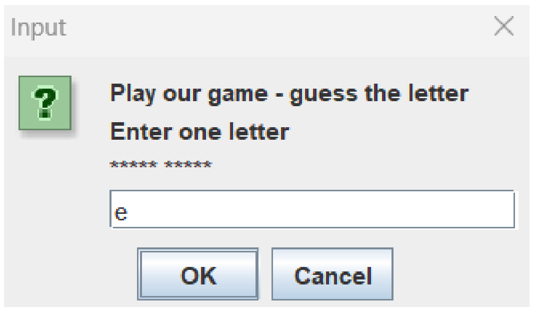
```
    for ( i from 0 to guesses.length )
        if ( guesses[ i ] = 0 )
            break loop
        else if ( guesses[ i ] = letter )
            return true
    return false
```

END *SecretPhrase*

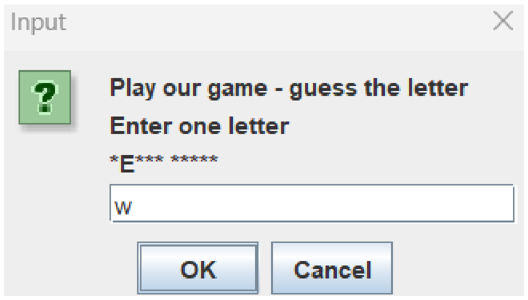
Test run(s):



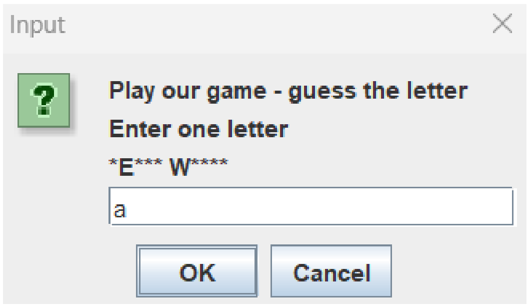
```
Input
Play our game - guess the letter
Enter one letter
***** *****

OK    Cancel
```



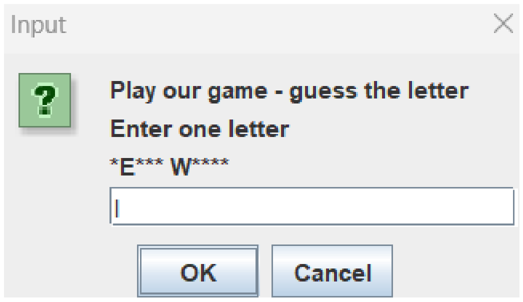
```
Input
Play our game - guess the letter
Enter one letter
***** *****
e
OK    Cancel
```



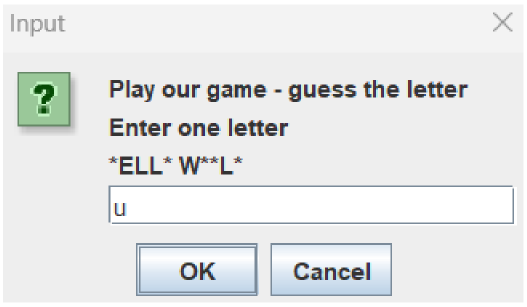
```
Input
Play our game - guess the letter
Enter one letter
*E*** *****
w
OK    Cancel
```



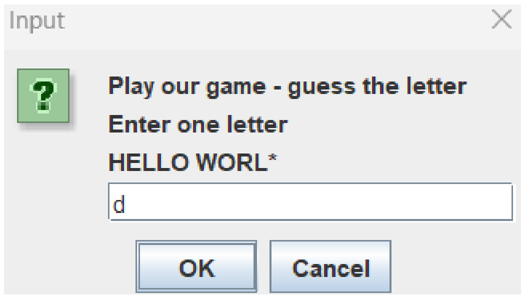
```
Input
Play our game - guess the letter
Enter one letter
*E*** W****
a
OK    Cancel
```



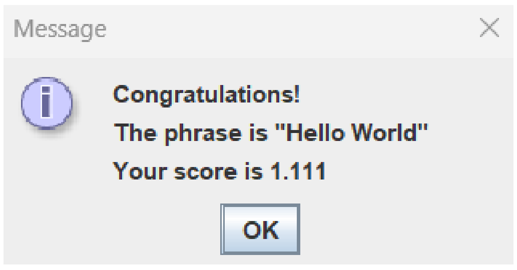
```
Input
Play our game - guess the letter
Enter one letter
*E*** W****
l
OK    Cancel
```



```
Input
Play our game - guess the letter
Enter one letter
*ELL* W**L*
u
OK    Cancel
```



```
Input
Play our game - guess the letter
Enter one letter
HELLO WORL*
d
OK    Cancel
```



```
Message
Congratulations!
The phrase is "Hello World"
Your score is 1.111
OK
```