Overview:

This program will take a stream for input. This stream contains some parameters to generate a puzzle. The program will try to find a solution for the given set of parameters and output the solution as a grid.

Data Structure to store input & output:

This program uses custom made data structure as well as built in data structure.

Variable => int noc [to store number of columns]
Variable => int nor [to store number of rows]
Variable => int now [to store number of words]

Spaces [Custom Made]

Object	Position	Row Position	Column Position	Length	Direction	Word
S1	0	0	0	5	H	Null
S2	1	1	2	5	H	Null
S3	2	1	4	5	V	Null
S4	3	4	3	4	V	Null

Space [ArrayList]

0	S1
1	S2
2	S3
3	S4

Words [Custom Made]

Object	Word	Used
W1	bash	false
W2	array	false
W3	frail	false
W4	plush	false

Word [ArrayList]

0	W1
1	W2
2	W3
3	W4

Grid [2D Character Array]

4		f				
3		r			b	
2		a	r	r	a	y
1		i			s	
0	p	l	u	s	h	
	0	1	2	3	4	5

Limitation:

1. The program will provide solution for less set of words.
2. For large data set [having a solution] if the counter crosses a certain value, there will be no solution.

Strategy & Algorithm:

- The program will store the input in above variables and data structure
- The loadPuzzle() function will invoke two methods, findIntersection() & createGrid()
- findIntersection() will find the intersections in the blank spaces in below form

Blank1	Index1	Blank2	Index2
0	1	2	4
0	4	3	3
1	0	2	2
1	3	3	1

  This means that 1st blank's (1st Index) & 2nd blank's (5th Index), 1st blank's (4th Index) & 4th blank's (3rd Index) have common letters and they intersect. These intersections are stored in an object of type intersections. These objects are stored in intersection.
- createGrid() will create a grid on blank spaces
- Once the above is set the Solve() function will be invoked to provide a solution
- I have used doWhile() loop to iterate and find a solution
- There is a boolean checkAll array which have initial value as false at the start of each iteration. The size of checkAll is the size of intersection.
- Now I iterate a for loop to access each blank space to enter a word in it
- To enter a word, I call a function getWord() to get eligible list of words. [eligible word list means the words which are not used in previous blanks]
- Now I will shuffle the word list to get different combinations of word and use the first word in the list to enter the blank space
- Set the status of word to true so that it is not used for the next blanks
- Now I will check letters at all the intersection points. If they are equal then I will set the boolean checkAll index to true.

- After checking all the intersection points the checkAll array will have boolean values [either true or false]
- If the array contains even a single ‘false’, then the solution is not found and the entire process will be done again
- If the array does not contain ‘false’, then the solution is found and then exits the doWhile loop
- After exiting the loop, fillGrid() function is called, which stores the letters in the proper positions
- After this the solution is printed when print() method is invoked.

Required functions and their return values:

1. public boolean loadPuzzle(BufferedReader stream)

This function will return **false** if:

- the stream is empty
- number of words does not equal to actual number of words
- number of columns is negative
- number of rows is negative
- number of words is negative
- column position is negative
- row position is negative
- length of word is negative
- direction is not ‘h’ or ‘v’
- word does not fit in the grid according to its length

if above conditions are not encountered then the function will return **true**

2. public boolean solve()

This function will return false if:

- the function is invoked before invoking loadPuzzle
- if no solution is found
- if word set is more (which has solution)

if above conditions are not encountered then the function will return **true**

3. public void print(PrintWriter outstream)

This function will print a solution if found.

This function will not print anything if:

- solution is not found
- the function is invoked before invoking loadPuzzle or Solve

4. public int choices()

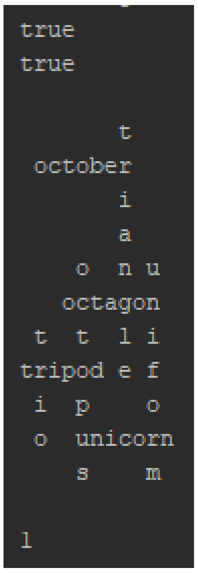
This function will return a value of **number iterations it took to get a solution**.

This function will return -1:

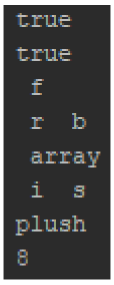
- solution is not found
- the function is invoked before invoking loadPuzzle or Solve

<u>Sample output of program:</u>

Success:

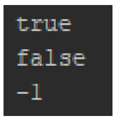


```
true
true

       t
 october
       i
       a
    o  n u
   octagon
 t  t  l i
tripod e f
 i  p    o
 o  unicorn
    s    m

1
```

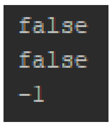


```
true
true
 f
 r  b
 array
 i  s
plush
8
```

Failure:



```
true
false
-1
```



```
false
false
-1
```