TCSS143
Fundamentals of Object-Oriented Programming-Theory and Application

LostPuppy

DUE: See Canvas Programming Assignment 5 link. **(DO NOT Zip your LostPuppy.java file. Your submission should simply be the LostPuppy.java source code file).**

For ALL programming assignments remember to obey all rules pertaining to Academic Integrity listed in the syllabus (DO NOT share code, use other people's code, OR post your solution on the internet).

Follow the Documentation Requirements as listed in the PDF file on Canvas. Do NOT use any constructs that have not been presented in the course thus far.

Overall, you will design a recursive method solution in an executable class source code file named ***LostPuppy.java*** that allows a puppy to find his way out of a maze at an exit point once he has entered it from an entry point.

Entry and exit points never occur at a corner of the maze, i.e. if a 2D array is used to store the maze and is named maze, entry and exit points will **never** occur at locations:

maze[0][0]
maze[0][maze[0].length -1]
maze[maze.length – 1][0]
maze[maze.length – 1][maze[0].length -1]

Facts:

- The walls of the maze are indicated with X's
- The paths are indicated with spaces
- The entry or starting point is indicated with an ' S '
- The exit point is indicated with an ' E '
- The input file is named "*MazeData***#***.txt*" where the **#** indicates a single numeric digit
- You can only open and read from the input file once
- The number of columns is unknown, but can be determined from the string length of the first line of input
- The number of rows is unknown, but can be determined by counting the number of lines read
- You can use any structures you want to help solve this assignment just as long as you use recursion and recursive backtracking to find the exit path.
- Though not required, a 2D array would intuitively be a possible means to store the characters representing the maze and if so:
  - ✓ Once the input data is read and stored, you can then instantiate the 2D array of char (as a hint, you might want to read lines into a string onto which you continually concatenate subsequent lines while counting each line read)
  - ✓ Place each character from the stored input data into the 2D array
- Place an asterisk '*' in each move location
- Place a minus sign '-' in cells where backtracking occurs

The following processes should not appear in main, but decomposed into separate methods (only suggesting method names and possible processing, but not requiring an exact naming or process):

getMaze:
Using the opened input Scanner, returns a structure to use containing the characters that make up the maze. If you apply the suggested method of continually reading and concatenating lines of input and chose to use a 2D array, while sequentially traversing a 2D array in row major order, you can use a separate index variable that is set to 0 before the loops begin and only increment after each character is copied from the String into the 2D array.

getStartExitLocation:
Once the array is filled, you need to locate the start position of the puppy when he first enters the building. Depending on how you are solving this assignment, you will probably want to replace the 'S' which indicates the start position, with the first asterisk '*'. Of course, this start position needs to be sent to the recursive doMaze method from the initial call.

You may also want to locate and return the Ending position (exit point containing the 'E') and replace the 'E' with a space. Depending on how you solve this, the exit location can be a possible base case for the doMaze recursive method. Alternatively, the E could stay in place as what is tested for the base case.

Up to you about everything the recursive method will need to know. However, since much of this data is primitive and methods can only return a single entity, you may need to consider creative means to pass so many primitives back to main so main can send them on to the actual recursive maze path solution method. One possibility is to store the row/column start and exit location values in 2-2 element arrays. One for the start and one for the exit positions. Again, these are all a matter you need to resolve and how you do it is up to you as long as you follow the basics:

- Do your own work
- See either me or the TA for assistance
- Decompose each operation/process into separate methods
- Never use class wide variables in an executable class (class that contains main), constants are fine

canMove:
Checks the given row and column indices for a move about to be made for validity, i.e. indices are within acceptable bounds of the array and the location in the array contains a space ' ' (or possibly the letter 'E' if you choose to use it for the base case). Returns a Boolean. This will be used by the recursive maze solution method just prior to a desired move.

doMaze (or some useful name):
Recursive solution to navigating the paths found within a given maze. Overall, needs to be passed the various data (including the maze) needed and returns false for each recursive call unless the exit location is reached which should set the variable used for the return statement to true.

Several approaches can be applied. However, fundamentally you need a:

Base Case: The currently passed Row and Column values indicate the maze's exit. Sets the return variable to true.

General (Recursive) Case: Based on the currently received row and column position, a move in 1 of 4 directions needs to be made. However, before making the next move, that move needs to be validated through a call to canMove. If canMove returns true, then the location to where the move will be made needs to be assigned an asterisk '*'. The move is then made by recursively calling doMaze by adding or subtracting 1 from one of the current indices based on the direction the puppy needs to proceed. If doMaze returns false, you need to assign a minus
sign '-' to the attempted location and you proceed to another of the remaining possible moves. Each recursive call should assign the returned value (true or false) to the local return variable which, in turn, is returned.

displayMaze:
Simply outputs the contents of the maze in a row major order.

main:
Declares the various variables needed that are passed between the various methods and opens the input file.
Calls the methods listed above as:

- getMaze
- getStartExitLocation
- doMaze: If returns false, display "No Path Found!", otherwise call displayMaze doe display on the console the maze and all paths the puppy took while looking for the exit.

Keep in mind, various data needs to be passed between the methods. This should only be accomplished by passing arguments to be received through the parameters listed in the method header and returning data using the return statement. Global variables should never be used as a means of sharing data between methods.

Though the maze solution may lend itself to less concerns of passing information to and from methods as an instantiable class with a constructor and class fields, this assignment is intended to help reinforce a CS major's ability to solve problems in a non-Object Oriented Programming design as with procedural languages which you will likely encounter during your career.

3 sample input maze files are included with these specifications in the provided assignment zip folder on Canvas.

DO NOT zip LostPuppy.java in a zip fold. Instead, simply submit your ***LostPuppy.java*** source code file to Canvas.

Sample Runs:

You won't be able to reproduce the "red" characters in your output. However, through the magic of word processing, I changed the font color of the path taken to red for ease of readability. As described above, the minus signs '-' indicate the backtracking that occurred and the asterisks reflect the final solution path.

Input File

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX                                  X
XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX X
XX              XX                    X
XXXXXXXX XXXXXX XXXXXXXXXXXXXXXXXXXXX X
X         X           XXX       X XXX X
XXXXXXXXXXXXXXXXXXXXX XXX XXXXXXX XXX X
X                           XX    X   X
X XXXXXXX XXXXXXXXXXXXXXXXX XXXXX X X X
X X       X        XXXXXXXX XXXXX X X X
X XXXXX XXX XXX XXXXXXXXXXX       X X X
X X     XXX XXX       XXXXXXXXX XX  X X
X XXXXX        XXXXXXXXX        XX XX X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XX X
XXX                               XXX X
XXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXX X
X      XXXXXXX XXXXXXXXXXXXXXX        X
XXXX XX                      XXXXXXXXXX
XXXX XXX XXXXXXXX XXXXXXXXXXXXXXXX   XX
XXXX     XXX                       X  X
XXXXXXSXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXEX
     Start                         Exit
```

Output Result

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX----------------------------------X
XXXX-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-X
XX--------------XX--------------------X
XXXXXXXX-XXXXXX-XXXXXXXXXXXXXXXXXXXXX-X
X---------X-----------XXX-------X-XXX-X
XXXXXXXXXXXXXXXXXXXXX-XXX-XXXXXXX-XXX-X
X---------------------------XX----X---X
X-XXXXXXX-XXXXXXXXXXXXXXXXX-XXXXX-X-X-X
X-X-------X--------XXXXXXXX-XXXXX-X-X-X
X-XXXXX-XXX-XXX-XXXXXXXXXXX-------X-X-X
X-X-----XXX-XXX-------XXXXXXXXX-XX--X-X
X-XXXXX--------XXXXXXXXX--------XX-XX-X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX--XX-X
XXX-------------------------------XXX-X
XXXXXXXXXXXXXX-XXXXXXXXXXXXXXXXXXXXXX-X
X------XXXXXXX-XXXXXXXXXXXXXXX--------X
XXXX-XX-**********           XXXXXXXXXX
XXXX-XXX*XXXXXXXX*XXXXXXXXXXXXXXXX***XX
XXXX--***XXX-----******************X**X
XXXXXX*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*X
```

`Hmmm... Looks like my algorithm did not do very well with these entry/exit points...`

`More Examples on the following page - - - >`

My algorithm does better on same maze above with a change in start and exit positions:

Input File **Start**

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXSX
XXXX                                  X
XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX X
XX              XX                    X
XXXXXXXX XXXXXX XXXXXXXXXXXXXXXXXXXXX X
X         X           XXX       X XXX X
XXXXXXXXXXXXXXXXXXXXX XXX XXXXXXX XXX X
X                           XX    X   X
X XXXXXXX XXXXXXXXXXXXXXXXX XXXXX X X X
X X       X        XXXXXXXX XXXXX X X X
X XXXXX XXX XXX XXXXXXXXXXX       X X X
X X     XXX XXX       XXXXXXXXX XX  X X
X XXXXX        XXXXXXXXX        XX XX X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XX X
XXX                               XXX X
XXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXX X
X      XXXXXXX XXXXXXXXXXXXXXX        X
XXXX XX                      XXXXXXXXXX
EXXX XXX XXXXXXXX XXXXXXXXXXXXXXXX   XX
XXXX     XXX                       X  X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Exit(see 3rd line from bottom on left)

Output Result

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*X
XXXX                                 *X
XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*X
XX              XX                   *X
XXXXXXXX XXXXXX XXXXXXXXXXXXXXXXXXXXX*X
X         X           XXX       X XXX*X
XXXXXXXXXXXXXXXXXXXXX XXX XXXXXXX XXX*X
X                           XX    X***X
X XXXXXXX XXXXXXXXXXXXXXXXX XXXXX X*X-X
X X       X        XXXXXXXX XXXXX X*X-X
X XXXXX XXX XXX XXXXXXXXXXX       X*X-X
X X     XXX XXX       XXXXXXXXX XX**X-X
X XXXXX        XXXXXXXXX        XX*XX-X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**XX-X
XXX           ********************XXX-X
XXXXXXXXXXXXXX*XXXXXXXXXXXXXXXXXXXXXX-X
X****  XXXXXXX*XXXXXXXXXXXXXXX--------X
X*XX*XX *******              XXXXXXXXXX
**XX*XXX*XXXXXXXX XXXXXXXXXXXXXXXX   XX
X-XX*****XXX                       X  X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Input File

```
XXXXXXXXXXXXXXXXXXXX
XX                 X
XX XXXXXXXXXXX X X X
X  X     XXXXX X X X
XX X X X     X X XXX
XX X X X XXXXX X   X
XX XXX X X     XX XX
X      X X XXXXXX XX
XXXXXXXX X   X     X
S        XXXXXXXX XX
XXXXXXXXXX         X
XX       XXXXXX XXXX
XXXXXXX         XXXX
XXX  XXXXXXX XXXXXXX
XX               XXX
XXXXXXXXXXXXXX XXXXX
X            X   X X
X XX XX XXXXXX XXX X
X  X X             X
XXXX XXX XXXXXXXXXXX
XX    X            X
XXXXXXXXXXXXXXXEXXXX
```

Start **EXIT**
(see row
10 above
On left)

Output Result

```
XXXXXXXXXXXXXXXXXXXX
XX***************  X
XX*XXXXXXXXXXX-X*X X
X *X  ***XXXXX-X*X X
XX*X X*X*    X-X*XXX
XX*X X*X*XXXXX-X** X
XX*XXX*X*X-----XX*XX
X *****X*X-XXXXXX*XX
XXXXXXXX*X---X   * X
*********XXXXXXXX*XX
XXXXXXXXXX     *** X
XX       XXXXXX*XXXX
XXXXXXX     ****XXXX
XXX--XXXXXXX*XXXXXXX
XX----------***  XXX
XXXXXXXXXXXXXX*XXXXX
X            X*  X X
X XX XX XXXXXX*XXX X
X  X X  *******    X
XXXX XXX*XXXXXXXXXXX
XX    X-********   X
XXXXXXXXXXXXXXX*XXXX
```

```
    Input File
                 EXIT
XXXXXXXXXXXXXXXXXXEX
XX                 X
XX XXXXXXXXXXX X X X
X  X     XXXXX X X X
XX X X X     X X XXX
XX X X X XXXXX X   X
XX XXX X X     XX XX
X      X X XXXXXX XX
XXXXXXXX X   X     X
X        XXXXXXXX XX
XXXXXXXXXX         X
XX       XXXXXX XX X
XXXXXXX         XX S
XXX  XXXXXXX XXXXX X
XX               X X
XXXXXXXXXXXXXXXXXXXX
               START 4th row
               From bottom
```

```
     Output Result

XXXXXXXXXXXXXXXXXX*X
XX--------------***X
XX-XXXXXXXXXXX-X*X-X
X--X-----XXXXX-X*X-X
XX-X-X-X-----X-X*XXX
XX-X-X-X-XXXXX-X** X
XX-XXX-X-X-----XX*XX
X------X-X-XXXXXX*XX
XXXXXXXX-X---X   * X
X--------XXXXXXXX*XX
XXXXXXXXXX       **X
XX       XXXXXX XX*X
XXXXXXX         XX**
XXX  XXXXXXX XXXXX-X
XX               X-X
XXXXXXXXXXXXXXXXXXXX
```