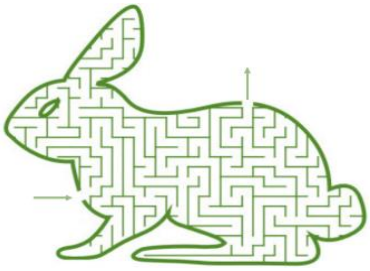
UNIVERSITY OF ALBERTA

Augustana Computing Science 112

Assignment #6 Maze Solver

Goals:

1. To be able to use predefined "official" Java Libraries (Stack, File, Scanner, some exceptions).
2. To correctly use a stack: pop, push, peek, size, empty.
3. To be able to read from a file, using File and Scanner.
4. To use an algorithm, and think about how that helps the programming process.
5. To have an *aMAZEing* time.



Reference:

Heise, Chapter 6

Instructions:

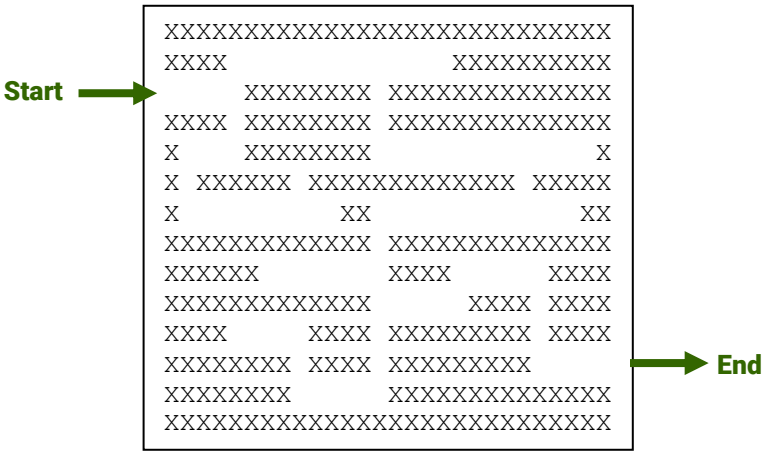
Complete the Java program, as per the specifications below. Work as an individual, and do *NOT SHARE* any lines of code with anyone else.

Document ALL of your code. Make sure that you add a file header (with your name, id #, date, and a summary). Also document each of the main steps in your solution. Ensure that all your variables are named descriptively. All methods should have a Javadoc header

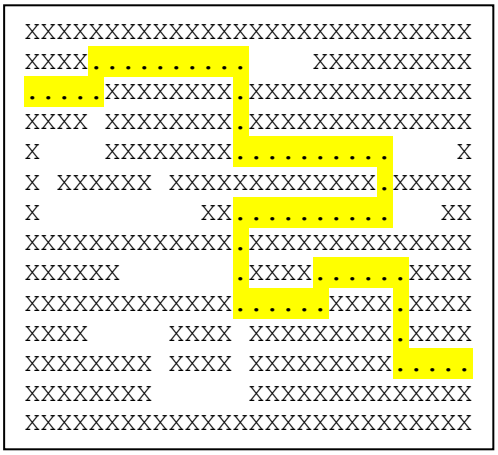
Programming style is also important. Eliminate useless steps, organize your solution and divide each problem into appropriate sub-problems, with their own methods.

The Program Specification:

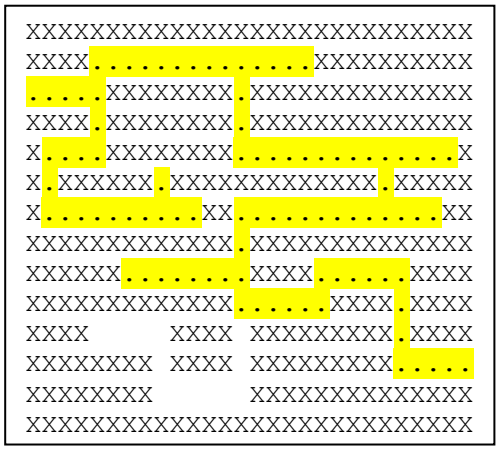
Construct a Java program that reads from a text file containing a maze, and then solve that maze, if possible. The text file will contain 'X's and blanks. The 'X's are walls (places you cannot step), while the blanks are hallways. You may assume that each line of the text file contains the same number of characters, but that particular number may vary between mazes (files). The starting position for the maze is on the left hand side and the ending position is on the right hand side. For example:



A maze will have only one starting position, and one ending position, though the positions may vary between mazes (files). Your program must find a path from the starting position to the ending position, or indicate that such a path cannot be found. Movement can only be up, down, left or right, not diagonally. The path must be shown with "cookie crumbs," i.e. dots. Here is the previous maze, solved (note that you would not put the color on the path, just the dots):



If the maze cannot be solved, print a message saying that. While printing the direct path is necessary to earn full marks (last 15%), if you can only print the maze with the path you followed, the non-direct path, that is still worth lots of marks, for example,



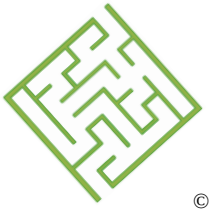
You must use Java's **`Stack<E>`** class in your solution, and use it well. This class should be imported:

```
import java.util.Stack;
```

Note that the Stack class is generic, which means that you will define it specifically to a type. Generic types must translate to a "class" type not a base (primitive) type. This means that you would use "`Character`" instead of "`char`", for example.

For file I/O, use a `Scanner` (just as with `System.in`) but now use a `File` object type. You must import the following:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```



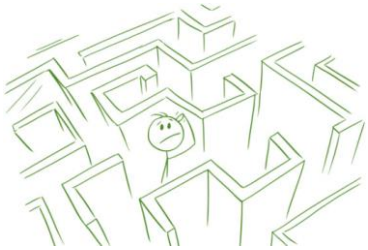
These four *given* `import` *statements are the only* `import` *statements allowed in your program. Stack exceptions should NOT be imported.*

Make your filename a **`public static String`** as the first thing in your class (so that we can easily change it to the file we want to read). Here is some sample code to show how to read from a file:

```
File myFile = new File(filename); //filename defined at top
try{ //file opens should be in try block to catch
     //exceptions, such as when file is not found
    Scanner input = new Scanner(myFile); //open file
    String aLine;
    int x;
    x = input.nextInt(); //read an integer
    aLine = input.nextLine(); //read an entire (remaining) line
}
catch(FileNotFoundException e){
    System.out.println("ERROR - File not found");
}
```

An Algorithm (marks path travelled, not direct path):

Here is an algorithm that will move through a maze. For full marks on this assignment, you will need to elaborate on this algorithm, but it is a great starting point, and if implemented well would earn 85%.



1. Read maze into a 2-d array
2. Make a stack
3. Find the start position and push that position on to the stack
4. While the stack is not empty do
    a. Pop a position.
    b. Store a dot in the array at the position.
    c. If the position is in the last column of the array then
        - There is a solution to the maze, so stop.
    d. Check in the cells adjacent to the position, and for each that contains a space, push that position onto the stack.
5. If we get here, there is no solution.

You may want to make a class for "positions," but since we only allow one `.java` class to be submitted, please make this a nested class.

Hand in on eClass:

1) One **`.java`** file.
2) A **`.pdf`** of your .java file
3) A **`.pdf`** of your test results on the file "`Maze3.txt`" from eClass/GitHub

Did you understand?

You should be able to answer these questions after completing the assignment. They are self-check questions - do not hand in your solutions.

1. What is a stack? Why was a stack the right data structure to use (vs. queue)?
2. What does push do? What does pop do?
3. When did you use peek?

4. Did you make an abstract data type to handle positions in the maze? Why or why not?
5. What did you do to your stack to enable you to undo moves that were incorrect?
6. Was it easier to write Java code when you had an algorithm? What might this say about writing code in general?
7. Where did you use an abstract data type?
8. What is encapsulation? Can you describe encapsulation, using a Stack as an example?
9. What is a generic data type? Where and how did you use a class that was defined with a generic data type in this assignment?
10. Where you always careful to ensure there was something on the Stack, before popping? What happens if you pop on an empty stack?

Why is it hard to get out of a maize maze?[1]

Why shouldn't you tell a joke in a corn maze?[2]

[1]You get corn-ered
[2]Because there are too many ears

Credits:
Bunny Maze: https://www.pinterest.com/pin/diy--309629346602639 8/
Stumped Dude: https://stock.adobe.com/ca/images/person-or-businessman-lost-in-labyrinth-or-maze-vector-cartoon-stick-figure-illustration/482587648
Plain maze: https://www.vecteezy.com/vector-art/3665064-maze-game-for-kids-funny-labyrinth-education-developing-worksheet-activity-page-puzzle-for-children-cute-cartoon-style-riddle-for-preschool-logical-conundrum-color-vector-illustration
Corn Winking:
https://www.google.com/search?q=winking+corn&rlz=1C1GCEA_enCA1005CA1006&oq=winking+corn&aqs=chrome..69i57j0i22i30j0i390l5.2800j0j7&sourceid=chrome&ie=UTF-8#imgrc=X21zRAQqIFsJWM
Corn Waving: https://www.freepik.com/premium-vector/cute-funny-corn-waving-hand-character_27069354.htm