CS 145: Homework 2

Maze

1 Description

Your primary goals in this homework are to learn about conditional statements and loops. You will do this in the context of writing a maze navigator.

You will write two classes for this homework: `Utilities` and `Maze`. Place them in the `hw2` package.

2 Introduction

Your software will load mazes in from a file and allow the user to move around them. Legal maze files will look something like this example:

```
4 3
2 0
1 2
**.*
*..*
*.**
```

The first line is the maze's width and height, second the starting location (0-based), and the third the ending location (0-based). The maze follows on the remaining lines, with asterisks representing walls and periods representing open paths. The origin—location (0, 0)—of the maze is the first character read.

The user wanders through the maze in a text-based console interaction. The maze is printed, with the player's location marked with '8'. A prompt is offered, at which the user uses the WASD keys, W to go north (toward the origin), A to go west, S to go south (away from the origin), and D to go east. The maze is reprinted and another prompt is offered. The game ends when the user reaches the ending location or types Q to quit. Here's the console interaction used to navigate the example maze:

```
**8*
*..*
*.**
> s

**.*
*.8*
*.**
> a
```

```
**.*
*8.*
*.**
> s

**.*
*..*
*8**

You escaped in 0.627 seconds!
```

3 Utilities

In order for us to work with the maze easily, you will need to shimmy it into a single `String`, a process called serialization. To serialize our maze, we'll just lay the maze lines end-to-end. The example maze above becomes "**.**..**.**" when serialized. To work naturally with the maze using Cartesian coordinate pairs, we'll need to convert Cartesian coordinates to `String` indices, a service which `Utilities` will provide. It has:

- A `static` method `cartesianToIndex` that takes four `int` arguments and returns an `int`. The first two arguments are the maze's width and height, and the last two are the (x, y) coordinate pair whose index we are trying to determine. If (x, y) are outside the 0-based bounds of the maze, -1 is returned. Otherwise, the index of the character at position (x, y) is returned. Draw some pictures to work out the formula for the conversion. Test this method before moving on.

4 Maze

Your `Maze` class serves as the main application class, though it defines a variety of helper methods that decompose the big task into smaller subtasks. Probably these helper methods should be `private`, but please make them `public` so our grading code can call them. All are `static`. It has:

- A method `print` that returns nothing and takes these arguments: the serialized maze as a `String`, the width of the maze, the height of the maze, the x-coordinate of the player's position, and the y-coordinate of the player's position. The last four arguments are all `int`s. It prints the maze line by line (deserialized), plotting '8' at the player's position. Test this method before moving on.
- A method `traverse` that returns nothing and takes these arguments: the serialized maze as a `String`, the width of the maze, the height of the maze, the x-coordinate of

the starting position, the y-coordinate of the starting position, the x-coordinate of the ending position, and the y-coordinate of the ending position. All arguments but the first are `int`s. This method lets the user walk through the maze. It places the user at the starting position, prints the mazes, prompts, and appropriately responds to the command. If the user enters Q, the method finishes. If the player enters W, A, S, or D, the user is moved, unless the move would position the user outside the maze or on a wall, in which case, some clear message like "You can't go that way!" is printed. These commands may be entered in either lower- or uppercase. If the user enters any other command, "Huh?" or some similar message is printed. This interaction continues until the user reaches the exit position or enters Q. If the user actually escaped the maze, the time it took to successfully navigate is printed. (Use `System.currentTimeMillis` to get clock readings.) Test this method before moving on.

- A method `traverse` that returns nothing and takes a single `String` argument, a path to a maze stored in a file. It reads the maze in from the file and call the other `traverse` method.
- A `main` method that either starts traversing a maze stored in a file. You can either hardcode the path or prompt the user to enter a path.

You are invited to share maze files and best times on Piazza.

5 Other Expectations

We need to be able to read your code. Some of your grade will be based on stylistic matters—not just a functioning final result.

- Variable and method names start with a lowercase letter and have capitalized internal words, e.g., `kittenCount`.
- Code should be documented. Explain what you are doing in brief statements.
- Code should be indented properly. Hit Control-Shift-F inside your Java files to let Eclipse clean up your code.

6 Extra Credit

3 points extra credit will be given if you provide a method `Maze.autotraverse` that takes a `String` argument for a maze file and automatically navigates (without user interaction) the maze. This code should not interfere with the main assignment code.

7 Submission

See `hw1` for submission instructions.