Homework: preparing the maze

Problem 1

Consider your basic maze. It has walls and pathways, and it has one (or more) starting point(s) and one (or more) finish point(s). Furthermore, one wall is just like another, and any open spaces (not including start and finish) are also identical. That sounds like a job for... enumerated types!

So, make an enumerated type `Square` that can represent those four values.

Each one has an associated one-character representation:

walls	`#`	(hash mark)
open spaces	`.`	(period)
start	`o`	(lowercase 'O')
finish	`*`	(asterisk)

Include in the enumerated type a `toString` method that returns a one-character-long string containing only the character corresponding to this `Square`. Also include a static method `fromChar` that returns the `Square` associated with the given `char`. (That is, a method with header `public static Square fromChar (char ch)` that, when provided with one of the four legal characters, returns one of the `Square` values. Any other input can generate an `IllegalArgumentException`.)

Problem 2

We've been talking about a five-operation *agenda* ADT for the last few classes, which is an important idea but not quite present in the Java libraries. In this problem, you'll adapt existing library code into a more abstract framework.

First, write the `Agenda<T>` interface. It should require five methods: `isEmpty`, `size`, `add`, `remove`, and `peek`.

Then, write two classes `MyStack<T>` and `MyQueue<T>` that implement that interface using, respectively, a LIFO policy and a FIFO policy. Code re-use is key here; all the real algorithm work is already done, and your only job for this problem is to *adapt* it for use in this framework.

Remember, you're using the Java libraries in this problem. Don't try to use `ListNode`!

Problem 3

Create a `Maze` class that can store the logical layout of a maze. It should contain a 2D array of either `Square` or `char`.[1] (Remember, a 2D array is just like a 1D array, except that you always provide two indices.)

`Maze` should have a constructor that can read from a `Scanner`. The format will be that the first line contains the dimensions of the maze (width and height), and subsequent lines each contain one row of the maze, with each character representing one square of the maze. BE CAREFUL about the interactions between line- and chunk-based scanning!

Also, write a `toString` method that makes a `String` representation of this `Maze` in the same format as the input. (This will be handy in testing your code...)

A simple example of the input/output format:

```
7 4
#######
#...#o#
#*#...#
#######
```

A more complex example:

```
12 10
############
#.#........#
#.#.######.#
#.#....#...#
#.###.*#.#.#
#...####.#.#
#.#.#..#.#.#
#.#.#.##.#.#
#o#......#.#
############
```

[1] For the purposes of this problem, it doesn't matter. If you had an easy time with `Square`, just use that—we'll be using that more later. If not, `char` is fine.