Project: solving the maze

All classes for this project are in package `maze`. You will be putting your existing classes from the previous homework in `maze` (or using the provided solutions) as well as writing several new ones. Note that this project does not explicitly list all of the classes you'll have to write—part of that will be design decisions left up to you.

The GUI

To handle all the fiddly graphical stuff, write a `MazeApp` class. When the GUI is started up (by typing `java maze.MazeApp`), it should present the following widgets across the top of the window:

- a text field
- a button labelled "Load"
- a button labelled "Start (stack)"
- a button labelled "Start (queue)"
- a button labelled "Step"
- a button labelled "Toggle animation"

When the user types a filename into the text field and clicks the "Load" button, the maze in that file should be read in and displayed in the applet: open squares are white, walls are black. The two start buttons should initiate the maze solution process (further described below). Once initiated, the step button makes the solution proceed by just one step, and turning on the animation uses a `Timer` (see below) to set the solution a-running at the rate of a few steps a second. (Clicking the animation button again should stop the animation.)

Somewhere conspicuous, the GUI should display a status line giving the current status of the solution process: "No maze", "Maze loaded", "Stack-based solution in progress", "Queue-based solution in progress", "Solution complete: finish reachable", "Solution complete: finish not reachable".

What we should see as the solution process goes on is that the squares that have been explored—yes, we can reach them, but no, we haven't found the finish yet—change from white to medium-grey. Whether stepping by hand or animating, we should see the maze slowly shaded in until the solver reaches the finish (or determines that it's unreachable).

Notes on `Timer`

There are two `Timer` classes in the java libraries, with similar functionality. You want the one in `javax.swing`, because the way it triggers its action is exactly the same way a `Button` does—via an `ActionListener`. Once the Timer is created, you will call its `start()`, `stop()`, and `isRunning()` methods to control it.

The one you don't want to use is in `java.util`. If you are freely importing everything from that package, make sure you're not accidentally using the wrong `Timer`!

The maze itself

From the previous homework, you have a `Maze` class that stores `char` or `Square` and can read from a `Scanner`. Before you get to the *solving* part, this project requires a `Maze` that can store `Square` values; if you did the `char` version or didn't complete that problem, you can either upgrade your previous work or use the published solutions as a starting point for this project.

You may assume that any well-formed maze will have exactly one start and exactly one finish. You may *not* assume that all valid mazes will be entirely enclosed with walls.

The agenda

As with `Maze`, it's expected that you've correctly solved the previous homework and have working `Agenda<T>`, `MyStack<T>`, and `MyQueue<T>` classes. If not, finish them or use the published solution.

The maze solver

The meat of the project will be the writing of a `MazeSolver` class (and associated classes), which will bundle up the functionality of determining whether a maze has a solution—that is, whether you can get from the start to the finish (without jumping over any walls). The algorithm one usually follows goes something like this: start at the start location, and trace along all possible paths to (eventually) all reachable open squares. If at some point you run across the finish, it was reachable. If not, it wasn't.

Boiling this down into pseudocode, we have the following:

At the start:

1. Create an (empty) agenda of locations to explore.
2. Add the start location to it.

Each step thereafter:

1. Is the agenda empty? If so, the finish is unreachable; terminate the algorithm.
2. Grab a location from the agenda.
3. Have we pulled this location from the agenda before? If so, no need to explore it again; this step is done.
4. Does the location correspond to the finish square? If so, the finish was reachable; terminate the algorithm.
5. Otherwise, it is a reachable non-finish location that we haven't seen yet. So, explore it as follows:
   - compute all the adjacent locations that are inside the maze and aren't walls, and
   - add them to the agenda for later exploration.
6. Also, record the fact that you've explored this location so you won't ever have to explore it again.

Note that this pseudocode is entirely agnostic as to what *kind* of agenda you use. You'll need to pick one when you create the agenda, but subsequently everything should work more abstractly in terms of the `Agenda` operations.

Design and discussion questions

In addition to the code you'll be writing for this project, you need to do a writeup that discusses the tradeoffs and design choices you had to make in the course of programming it. You would do well to read the following questions *first* and think about them early, and take notes on what you plan to do—they will help you shape your design. There are also a few hints as to what you can and can't do.

The writeup should not just be a bullet list of answers to the following questions. It should likewise not be a disconnected sequence of sentences that answer the questions. This is essentially a lab report, and as such it should be in clear prose. That said, you should make sure that you do address all the issues raised in the questions.

What kind of a `Maze` will the `MazeApp` have before you've loaded any file?

Which object should be responsible for generating the status message?

How will you share the responsibility for actually drawing the maze? Remember that the solution-in-progress should display explored squares in grey—which data structure is keeping track of that information?

How will you represent locations? Why can't you just modify `Square` for this?

How will you keep track of the locations you've explored vs those you haven't? There's more than one right answer—be prepared to discuss the tradeoffs involved.

What other data structure(s) would you need if your solver needed to be able to report the actual path from the start to finish, instead of just the fact that one existed (or not)?

Whose responsibility should it be to actually create the `MyStack` or `MyQueue`, as appropriate?

The obvious time to click the Step button is immediately after one of the Start buttons, which was probably clicked right after the Load button. What should happen if a user clicks buttons at an unexpected time? (Hint: *not* a `NullPointerException`.)