EECS 397

Software Craftsmanship

2013 Fall Semester

Programming Assignment 2

Due at your recitation session on September 9-13, 2013

Reading

Read Chapters 14, 15, and 16 in the textbook.

Programming

A lab mouse is being trained to escape from a maze. The maze is made up of cells, and each cell is connected to some other cells. However, there are obstacles in the passage between two cells and therefore (i) it takes some time to go through the passage, and (ii) the passages only allow the mouse the go one way, but not the other way around.

First, as a general rule, if a passage has a passage time equal to `Integer.MAX_VALUE`, then in practice the passage is unusable and the mouse will not be able to cross it to reach another cell.

The class `MazeCell` represents a cell in the maze and its passages to adjoining cells. The cell is marked as invalid when it is first constructed, and it turns to a valid cell once it is connected to other cells. `MazeCell` implements the following methods:

- `MazeCell()` sets up an invalid `MazeCell`. The constructor must not throw any exception.
- `boolean addPassages(Map<MazeCell, Integer> passages)` sets up the outgoing passages from this `MazeCell`, and makes this `MazeCell` valid. The argument gives the travel time along a passage from this `MazeCell` to another `MazeCell` in seconds. There can be only one passage between one cell and another one. A travel time equal to `MAX_VALUE` represents an impassable connection. The travel time can be zero or negative (this will be fixed in future assignments). If this `MazeCell` had already been connected with some passages (that is, this `MazeCell` was already valid), the method does nothing and returns `false`, otherwise it returns `true`.
- `boolean isValid()` returns whether this `MazeCell` is valid.
- `int hashCode()` returns a hash code for this `MazeCell`.

- `Map<MazeCell, Integer> passages()` returns the passages from this `MazeCell`, which must comprise all other `MazeCells` whose travel time is less than `MAX_VALUE`. You should take steps so that the returned map cannot be abused to change the connections from this `MazeCell`. The method throws a `UninitializedObjectException` (see later) if this `MazeCell` is invalid.
- `Integer passageTimeTo(MazeCell cell)` gives the times to cross from this cell to the given cell by traversing a single passage. If there is no direct passage, `passageTimeTo` returns `MAX_VALUE`. The method throws a `UninitializedObjectException` if this `MazeCell` is invalid.
- `Set<MazeCell> connectedCells()` returns a set of cells to which this cell is directly connected. The returned set must comprise all other `MazeCells` whose travel time is less than `MAX_VALUE`. You should take steps so that the returned set cannot be abused to change the connections from this `MazeCell`. The method throws a `UninitializedObjectException` if this `MazeCell` is invalid.
- `boolean isDeadEnd()` returns whether this cell is a dead end (it has no valid outgoing passages: `connectedCells()` is empty). The method throws a `UninitializedObjectException` if this `MazeCell` is invalid.
- `String toString()` returns a human readable code for this cell (but no information on neighboring cells).

You must create the `UninitializedObjectException` class as a subclass of Exception with the four standard constructors:

- `UninitializedObjectException()`,
- `UninitializedObjectException(String message)`,
- `UninitializedObjectException(String message, Throwable cause)`, and
- `UninitializedObjectException(Throwable cause)`.

A special cell will represent the external world out of the maze. The external cell is valid but has no passages to any other cell (the mouse won't be able to get back into the maze). In practice, the external world is undistinguishable from a dead end.

A `MazeRoute` represents a path through the maze:

- `MazeRoute()` is the constructor that initializes this route to an empty (a route with no cells) and invalid route.
- `boolean addCells(List<MazeCell> route)` adds the given list of cells to the route and makes this route valid. It throws an `UnitializedObjectException` if any cell is invalid. It returns `false` if the `MazeRoute` already contains some cells (if it was already valid), `true` otherwise.
- `boolean isValid()` returns whether this `MazeRoute` is valid.

- `List<MazeCell> getCells()` returns the list of cells in the maze. You should take steps so that the returned list cannot be abused to change the route. The method throws a `UninitializedObjectException` if this `MazeRoute` is invalid.
- `String toString()` returns a human readable representation of this route, with a code for each cell along the path and the travel time from one cell to another. If there is no passage (or the passage time is `MAX_VALUE`), a special string (like "no passage") should be given instead. The method throws a `UninitializedObjectException` if this `MazeRoute` is invalid.
- `Integer travelTime()` returns the time needed to follow this route. It returns 0 if the route consists of only one cell, `MAX_VALUE` if the route contains an inexistent or impassable passage. The method throws a `UninitializedObjectException` if this `MazeRoute` is invalid.

Additionally, these classes may contain as many auxiliary private methods as you see fit, and additional helper classes may be defined.

Your classes must be documented in JavaDoc. Test cases must be created in JUnit to test all its public methods.

Discussion Guidelines

The class discussion will focus on:

- High-level code organization
- The design and documentation of the implementation
- Straight-line code, conditional, and loops

On-Line Submission

Bring a copy to recitation to display on a projector. Additionally, submit an electronic copy of your program to blackboard. Turn in with your code:

- A README file explaining how to compile and run the code.
- A Make, Ant, or Maven build file.
- A comment at the top of the file containing your name, email address, a one-sentence description of the file, and if necessary a longer comment describing the design of the file.

The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted.