EECS 397

Software Craftsmanship

2013 Fall Semester

Programming Assignment 3

Due at your recitation session on September 16-20, 2013

Reading

Read Section 17.1, pages 403-404 ("Rewrite with a status variable"), and Chapter 19 in the textbook.

Programming

Implement the `Maze` class, which represents a set of cells with their passages between them:

- `Maze()` is the constructor that initializes this `Maze` to an empty (a maze with no cells) and invalid maze. It must throw no exceptions.
- `boolean addCells(Set<MazeCell> cells)` adds the given set of cells to the maze and makes this maze valid. It throws an `UnitializedObjectException` if any cell is invalid. It returns false if the `Maze` already contains some cells (if it was already valid), `true` otherwise.
- `boolean isValid()` returns whether this `Maze` is valid.
- `String toString()` returns a human readable representation of this maze: for each cell, the string should represent the neighboring cells and the travel time through the direct passages. The method returns the String "Uninitialized Maze" if this maze is invalid.
- `MazeRoute routeFirst(MazeCell initialCell)` returns a route within the maze starting from and including the given initial cell assuming that in each cell the mouse chooses an arbitrary passage. The mouse should never cross a passage whose travel time is `MAX_VALUE`. The route should stop if the mouse exits the maze, reaches a dead end, or reaches a cell that it had previously visited. It returns an empty list in case of error, for example if at any point the mouse finds itself in a cell that does not belong to the maze (either at the initial cell or throughout the route). The method throws a `UninitializedObjectException` if this `Maze` is invalid.

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

In this programming assignment, you will revisit the previous programming assignment (`MazeCell` and `MazeRoute`) to accomplish two additional objectives: reduce the McCabe complexity of your software and its use of non-structured programming.

Discussion Guidelines

- High-level code organization
- The design and documentation of the maze implementation
- Functions that exceed McCabe's complexity of 4 (if any)
- Non-structured programming constructs and break statements (if any)

On-Line Submission

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 over 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.