EECS 293

Software Craftsmanship

2013 Fall Semester

Programming Assignment 4

Due at your recitation session on September 23-27, 2013

Reading

Read Chapter 5 in the textbook.

Programming

First, make any changes that were discussed in your recitation session.

A common error handling technique is to use a status variable (for example, pages 403-404: "Rewrite with a status variable"). In this assignment, you will rewrite `MazeCell` so that it prevents time values that are zero or negative, and uses a status variable to communicate potential errors. First, define a class `Status` nested within `MazeCell`:

- `Status` contains a public enumerated type called `Code` with values `OK`, `ALREADY_VALID`, `INVALID_TIME`. Each code has an associated `String` description that can be retrieved with a method called `getMessage`. A tutorial on Java enums is posted on blackboard.
- `Status()` a constructor that initializes the status to `OK`. It must throw no exceptions.
- `void set(Code code)` and `Code get()` to set and get the current status.

Then, rewrite `MazeCell` with:

- `void addPassages(Map<MazeCell, Integer> passages, Status status)` sets up the outgoing passages from this `MazeCell`, and makes this `MazeCell` valid. The first argument gives the travel time along a passage from this `MazeCell` to another `MazeCell` in seconds. A travel time equal to `MAX_VALUE` represents an impassable connection. If this `MazeCell` had already been connected with some passages (that is, this `MazeCell` was already valid), the method does nothing but it sets `status` to `ALREADY_VALID`. If any travel time is zero or negative, the `status` will be `INVALID_TIME`. Otherwise, the `status` will be `OK`.

Since different lab mice may take different lengths of time to clear the obstacles from one cell to another, the passage time can be random. Specifically, suppose that the passage time can vary randomly between 1 second and the value given in the `MazeCell`. An

introduction to random numbers in Java is posted on blackboard. Implement the following additional `MazeRoute` method:

- `Integer travelTimeRandom()` returns the time needed to follow this route assuming that on each passage the mouse takes a random time between 1 second and the value given in the corresponding `MazeCell`. However, if the route contains an impassable (one with `MAX_VALUE` travel time) or invalid passage, it returns `MAX_VALUE`. It returns 0 if the route consists of only one cell. The method throws a `UninitializedObjectException` if this `MazeRoute` is invalid.

Moreover, implement the following additional `Maze` method:

- `MazeRoute routeRandom(MazeCell initialCell)` returns a route within the maze starting from and including the given initial cell assuming that in each cell the mouse chooses a random passage (if a cell has *n* outgoing passages, the mouse will choose one of the *n* passages at random). 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.

Create a test case where `routeRandom` can produce a different path than `routeFirst` even if started from the same `initialCell`, and an example where the two methods must always produce the same path.

As usual, 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 your project
- Functions that exceed McCabe's complexity of 4 (if any)
- Non-structured programming constructs and break statements (if any)
- The difference between an arbitrary choice and a random choice.

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 that works with both Java.
- 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.