Coursework II: Maze

Where should I start?

In this coursework we're asking you to implement a JavaFX application that programmatically solves, and visualises, mazes.

1. Before you Take Your First Steps

It's really critical that you understand what exactly you'll need to implement. Make sure to read the full instructions document carefully. You can also download a large version of the UML diagram.

2. First Steps

1. You'll be working in a new gitlab respository for this coursework. Clone a copy to your local machine and take a look at the layout. Make sure you understand where the Java files should be placed (`src`) including the existing `MazeDriver.java` file. Create a new (empty) Java source file `Maze.java` -- this file will ultimately correspond to the `Maze` class shown on the UML diagram. Take a look at the two example maze files `resources/mazes/maze1.txt` and `resources/mazes/maze2.txt`). These are all the files you'll need to make your first steps. **Be sure to regularly commit your changes and push them back to the gitlab hosted respository.**

2. We suggest that a good place to start is with reading the text files in using the `Maze` class. Outline your `Maze` class with one empty method `fromTxt` to `Maze` according to the specification described in the UML file. Implement `MazeDriver` so that it creates a new `Maze` instance and then calls `from Txt` on that instance. You should pass the path to one of the downloaded text files as the parameter for `from Txt` .

3. Revisit your code from Lab 5, Exercise 1 (step 8) or Section 18.5 of the book. Can you get the `fromTxt` method to read the text files line by line and print the result?

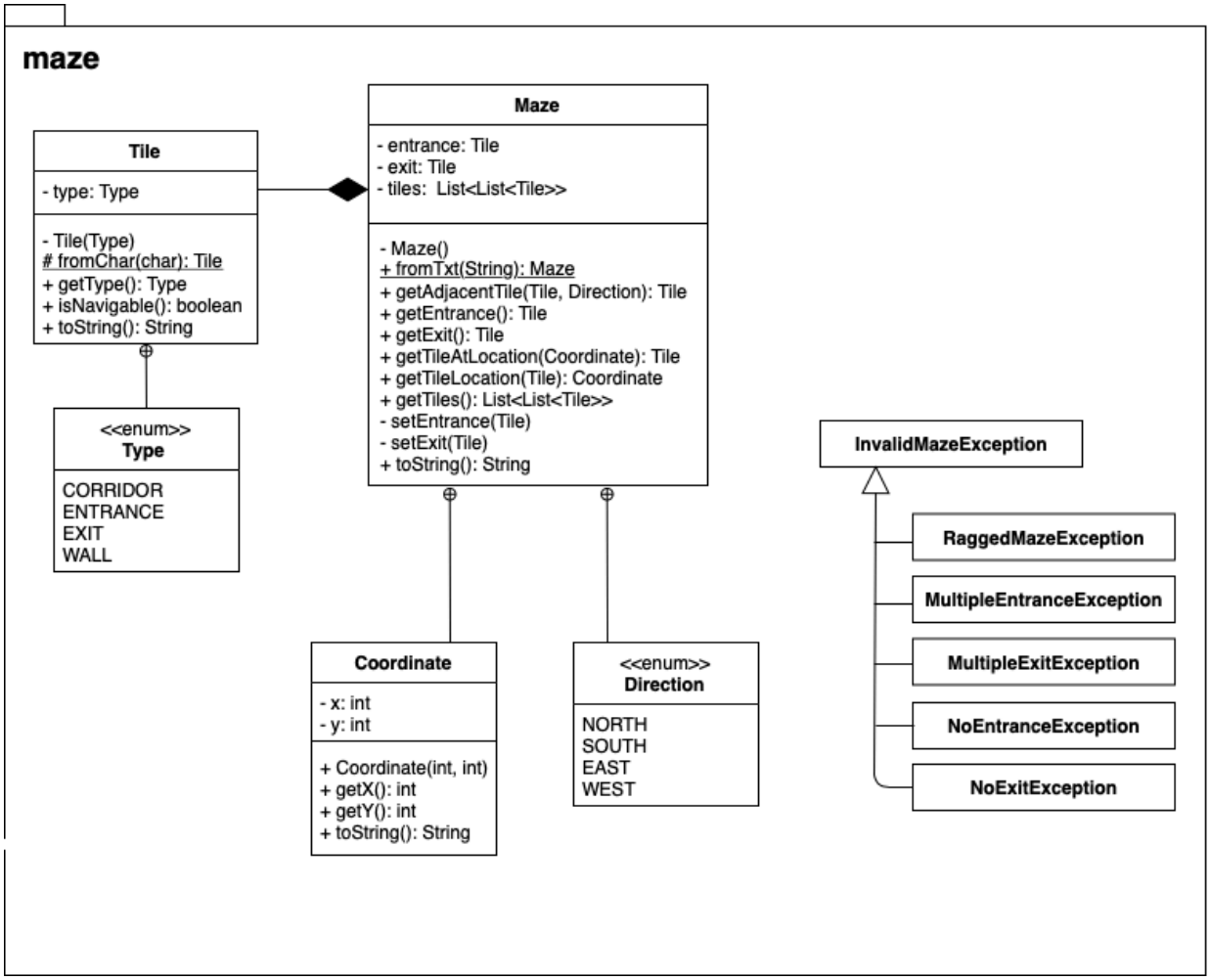
4. Before you can get much further you'll need to create the `Tile` class and the enum `Tile.Type` . `Tile` is a fairly simple class, the only thing likely to be new here is the idea of nesting an `enum` inside the `Tile` class itself -- we'll do more on nested reference types in `Week 7` but in this case all you need to do is declare the `enum`

as you would normally but make sure the definition is inside the `Tile` class.

5. Now that you have a `Tile` class, you should be able to fully-implement `fromTxt` (which probably also involves implementing the constructor, `setEntrance` and `setExit` and many of the `InvalidMazeException` classes) and `toString` (which probably involves implementing `getTiles`. Remember that you can add extra methods to `Maze` if they help you with your implementation. (However, you must not change the signature of any method defined in the UML.)

6. Take another look through the `maze` package UML. Implement any remaining features and test them using your `MazeDriver`.

7. Read Chapter 11 of the book. Add Javadoc to everything you've written so far.



3. Next Steps

1. Read Chapter 19 of the book. Move all the code you've developed so far except for `MazeDriver.java` into a folder called `maze` and add the package statement to the top of each class file. Add import statements to any class files that need them (including `MazeDriver.java`). Check that the driver still runs and creates mazes correctly.

2. Create a new folder `maze/visualisation` -- this will contain the code for the `maze.visualisation` package. Take a look at the material from Week 8 and start thinking about how you're going to visualise your maze. What components might you need? Make sure you add Javadoc for your new classes.

3. Create a new folder `maze/routing` -- this will contain the code for the `maze.routing` package. Take a look at the specifications for classes in the `maze.routing` package, can you make a start on some of these? Make sure you add Javadoc for your new classes.