# Maze Library Emulator

## Repo Structure:

Packages:

* Main (DO NOT MODIFY):
  + This package contains the main class.
  + Main creates an instance of a solver and calls solveMaze().
* Maze (DO NOT MODIFY):
  + This package contains all code for generating and moving.
  + This package includes the Maze, Coordinate, and Direction class.
* Solver (MODIFY):
  + This package is where your maze navigating code will go.

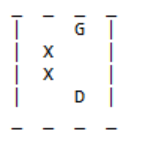
## Public Coordinate Functions:

* int getX()
  + Returns the x portion of the coordinate pair.
* int getY()
  + Returns the y portion of the coordinate pair.

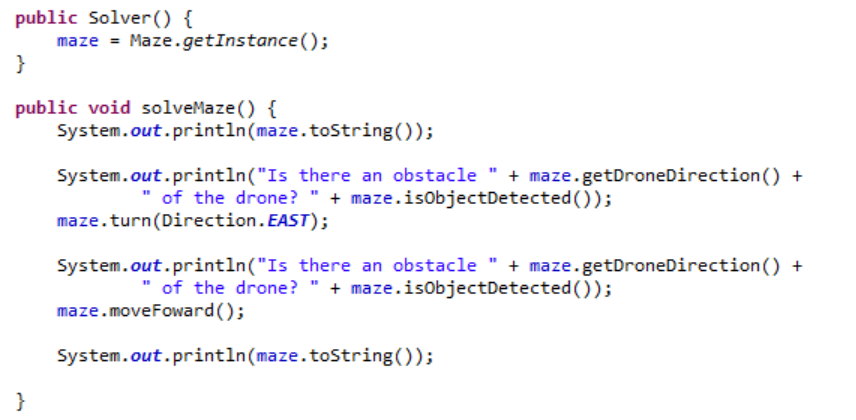
## Direction Enum Values:

* NORTH
* SOUTH
* EAST
* WEST

## Public Maze Functions:

* Maze getInstance()
  + Returns an instance of the ‘Maze’
  + Maze is a singleton class, meaning every time you call getInstance(), it will return the same maze instance.
  + Each time you run the maze program, a different maze will be preloaded. This is so you cannot hard code your way through the maze.
* Coordinate getDroneCurrentCoordinates()
  + Returns a Coordinate containing the drones current (x,y) location in the maze.
* Direction getDroneDirection()
  + Returns a Direction enum. This is the direction that the drone is currently facing.
* Coordinate getGoalCoordiantes()
  + Returns a Coordinate containing the goals (x,y) location in the maze.
* boolean isObjectDetected()
  + Returns a boolean value. Returns true if there is an obstacle in front of the drone in the direction that the drone is facing.
  + Returns false if there is no obstacle in front of the drone in the direction the drone is facing.
* void moveForward()
  + Moves the drone forward one space in the direction that the drone is facing.
  + Program will exit if the drone moves into an obstacle.
* void turn(Direction)
  + Will turn the drone to the direction passed.
  + If the drone is currently facing south and then is asked to turn north, the new current direction of the drone will be north.
* String toString()
  + Returns a String of the Maze’s current state
  + Example:
    - 
      * X = Obstacle
      * D = Drone
      * G = Goal

## Example Code Usage



Resulting Output:

