Flying Around

In a forest there are hawks, rodents and wander-stones. The hawks have wander-stones as their beloved toys (because wander-stones are hard and have an attractive shape), but they are also tempted by the rodents which they like to grab and eat. We are in possession of a situation map of the forest, with special marking for important elements. Here are the details:

1. **Hawks**, marked with capital **H** letters try get a wander-stone to play with, and will normally go to grab the closest wander-stone. But if they see a rodent which is closer, they will be attracted by it, and fly toward the nearest rodent they saw to grab it. A hawk flies 1 square row per step in one of the directions N, NE, E, SE, S, SW, W, NW. If a hawk grabs a rodent, both the hawk and the rodent will disappear from the forest. If it grabs a wander-stone, the wander-stone will disappear, and the hawk will stop moving, enjoying the wander-stone.
2. **Rodents,** marked with capital **R** letters, will move towards the nearest wander-stone. When they are close, (like the rodent near wander-stone **5**, or the rodent near wander-stone **C**, in Fig. 1,) they do not move. A rodent moves 1 square per step in one of four possible directions N, E, S, W.
3. **Wander-stones**, symbolized with hex digits (from `0` to **F**, only uppercase letters are used), can be caught by the hawks, and they also attract rodents. They do not move at all and disappear once grabbed by a hawk.

Everything happens in *steps*.

- A step is taken when the user presses the 'Enter' key. First, the hawks move, and then the rodents.
- The process ends when there are no more tempting rodents, or every hawk has grabbed a wander-stone and stopped, or there are no more wander-stones to grab.

At the beginning of the game, and after each step you should print to `System.out` – *text mode* – the current situation map – as in Figure 1 below.

There is a package provided, called `oop.forest` which contains in class `Given` the following public static methods:

- `Coord[] readAny(char[][] map, char symbol)` returns an array with coordinates for the specific symbol character.
- `Coord[] readWanderStones(char[][] map)` returns an array with wander-stone coordinates.
- `Coord[] getNearest(Coord me, char[][] map, char type)` returns an array of closest coordinates to `me` sorted in ascending order. The `type` parameter can be '**R**' for rodent or '**W**' for Wander-stone. If no elements of that type exist, it returns `null`.
- `Coord findNewStep(char[][] map, char myType, char targetType, Coord myPos, Coord targetPos)` returns the coordinate of the move position towards the position `pos`, for an object with type symbol `myType` at position `myPos`, executable in one step. The `targetType` parameter can be '**R**' meaning for rodent or '**W**' for Wander-stone. The `myType` parameter can be '**R**' for rodent or '**H**' for hawk.

`Coord` is an instantiable class also included in the package `oop.forest`, with integer fields named `row` and `column` and a double field named `distance` setters and getters for those fields. Rows and columns are numbered starting at zero. It has two constructors, one without parameters, which sets everything to zero, and one with two parameters: `Coord(int row, int column)`.

You should use what is provided in the package in your implementation, not implement them yourself.

An example start configuration, shown in Fig. 1, contains the size of the forest, and the situation map. The start configuration is read from a file named `FirsHR.txt`.

Draw the *class diagram* and develop a *java standalone* program to simulate this game. Don't forget to briefly document it.

```
14 84
....................................................................................
..H..............................................................7..................
..........................................................................8.........
..........................................................6.........................
..........0.......1....................5............................................
...................2....................R...........................................
.................................................................R...............9..
.........H..............R....4.....................................................
....................................................................................
...............R..........3.....................R......................A............
...H............................................R...................................
......................................H....................B........................
......................................H.............................................
```

Fig. 1 Example of a possible start configuration. The map does not include first line.