Rover 3-4

Refactor your **Rover** class to model one instance of a robotic rover, with its own position and direction. This is mostly a matter of converting all of your **static** fields and methods to be "non-static" a/k/a instance fields and methods; and then reworking your **main** method to create an instance of the class, and to call the same methods it did before, but now on that instance instead of on the class itself.

When that's working, add a field and a few more methods that provide a different way of directing the rover. The rover will now keep track of its own **commands** string, which starts as the empty string and can be extended by calling a method **receiveCommands**. This might be called multiple times, and each time the given string of commands should be appended to **commands**. This method should also convert a digit into that many 'M' commands – so '5' becomes "MMMMM" for example.

Add a method **takeNextStep** which executes just the first command in the string, and removes it from the string. (This is a lot easier since you've normalized the command string to be all individual letters.) This will allow the caller to move the rover around one step at a time, for example on a timer kept by the caller.

Add a method **isBusy** that returns **true** if the rover currently has any commands to process, **false** if it is out of commands and hence idle. Make **takeNextStep** start out by checking if the rover **isBusy**, and if not, don't try to process the command string – since it's empty and processing would fail!

Also provided in this folder, **Rover3**. are some test programs. Copy these into your project, and integrate your **Rover** class to work with them. You'll need to move your class into the package **com.amica.mars**, for starters, and make sure that your method names and parameter signatures align with the expectations of these programs. Among other things this will require that you add some "getter" methods to make the rover observable by the test programs. Note in particular the need for a **getDirection** method, and that this method should return a **Direction**.

Test program 1 exercises the new interface, adding a command string and running the rover until it's no longer busy. You can try each of the command strings there – two are commented out, but you can swap them in and test each one. Expected final positions and directions are:

```
4L2R2L1 -> (4, 3) and facting east
LL4R2R1L2 -> (-4, -3) and facing west
R4LL2R17 -> (2, 8) and facing north
```

Test program 2 proves out the ability to model multiple rovers that can move concurrently. The last lines of output should be:

directions are:

```
Rover 1: The rover is now at (4,3), and facing east.
Rover 2: The rover is now at (-5,2), and facing west.
```

And the **TestUI.java** will let the interactive user issue commands to each of two rovers, which then will execute pending commands, one second at a time, and move visibly about a 13x13 grid.