KATALYST BROWSE ABOUT

# Mars Rover

**Difficulty:** Competent

**Estimated Duration:** 2 hours 10 minutes

## Summary

It's very common to find programs which rely on state. This is especially true of user interface code.

However, state is also a very common source of bugs. The main reason for this is that state is simply hard to reason about.

To a certain extent, good test cases help in stateful programs. However, we find that testing on its own is not enough: a good design is essential too.

This kata lends itself to a variety of object-oriented design approaches, which makes it a great way to practice your design skills. It is well-suited to two Design Patterns in particular: the Command pattern and the State pattern.

There are a variety of ways to practice with this kata. We suggest making an initial implementation first, following the SOLID principles and the 4 rules of simple design. Afterwards, see if you can refactor your solution to use either of the design patterns above. You could also retry the kata from scratch with a particular solution in mind.

## Instructions

A squad of robotic rovers are to be landed by NASA on a plateau on Mars.

This plateau, which is curiously rectangular, must be navigated by the rovers so that their onboard cameras can get a complete view of the surrounding terrain to send back to Earth.

Your task is to develop an API that moves the rovers around on the plateau.

In this API, the plateau is represented as a 10x10 grid, and a rover has state consisting of two parts:

- its position on the grid (represented by an X,Y coordinate pair)
- the compass direction it's facing (represented by a letter, one of `N`, `S`, `E`, `W`)

## Input

The input to the program is a string of one-character move commands:

- `L` and `R` rotate the direction the rover is facing
- `M` moves the rover one grid square forward in the direction it is currently facing

If a rover reaches the end of the plateau, it wraps around the end of the grid.

## Output

The program's output is the final position of the rover after all the move commands have been executed. The position is represented as a coordinate pair and a direction, joined by colons to form a string. For example: a rover whose position is `2:3:W` is at square (2,3), facing west.

## Obstacles

The grid may have obstacles. If a given sequence of commands encounters an obstacle, the rover moves up to the last possible point and reports the obstacle by prefixing `O:` to the position string that it returns. For instance, `O:1:1:N` would mean that the rover encountered an obstacle at position (1, 2).

## Examples

- given a grid with no obstacles, input `MMRMMLM` gives output `2:3:N`
- given a grid with no obstacles, input `MMMMMMMMMM` gives output `0:0:N` (due to wrap-around)
- given a grid with an obstacle at (0, 3), input `MMMM` gives output `O:0:2:N`

## Interface

There are no restrictions on the design of the public interface. In particular, much of the details of the obstacle feature's implementation are up to you.

A public interface to the API could look something like this:

```
class MarsRover {
    public MarsRover(Grid grid);
    public String execute(String command);
}
```

Rules:

- The rover receives a char array of commands e.g. `RMMLM` and returns the finishing point after the moves e.g. `2:1:N`
- The rover wraps around if it reaches the end of the grid.

Credit: Google Code Archive

## Useful Links

### Books

- Head First Design Patterns by Eric Freeman et al.
- Understanding the Four Rules of Simple Design by Corey Haines

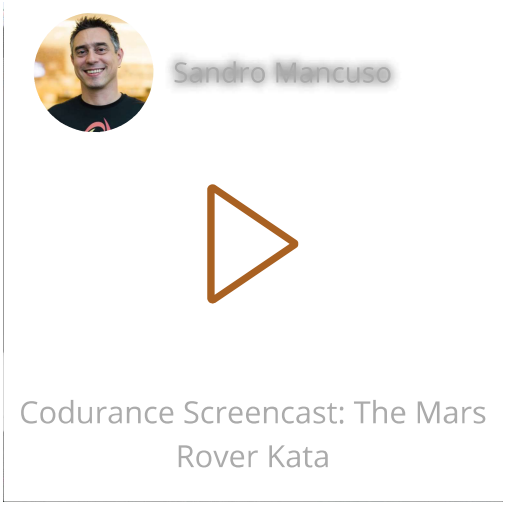
### Articles

- Mars Rover Kata: Refactoring to Patterns by Simion Iulian Belea

### Solutions

- Java by Sandro Mancuso

### Videos




SOLID design patterns design starter

Psst!

Katalyst is in beta and we'd love your feedback.

Email us at katalyst@codurance.com and let us know what you think.

## We are hiring

Are you looking for autonomy, mastery and purpose in your career? We are looking for people that share the same values of pragmatism, professionalism and transparency that we do.

Learn more


Software is our passion.

We are software craftspeople. We build well-crafted software for our clients, we help developers to get better at their craft through training, coaching and mentoring, and we help companies get better at delivering software.

Company Registration No: 8712584