This shows a reusable and thread-safe design of the problem in question. The problem specifies how the rover should move with the provided commands.

However, as in a real-life scenario –

* There might be a need to change the Rover’s movement strategy at any time in future
* Also, there might be a need where after a Rover has been placed on Mars, the controller from earth might need to deploy new movement strategy to an existing rover

These are not hypothetical questions, as similar change in requirements can happen in any software design. It’s important that the design should show resilience/flexibility in face of future challenges.

Keeping in mind the above arguments, the components have been designed with the following goal in mind –

* Prefer composition over inheritance
* All Rover instantiation should happen through a Rover Factory
* Implement a strategy pattern, where –
  + Any new rover is instantiated with a default Movement algorithm (Embedded inside the Rover class). However, while creating a rover, the factory injects a specific movement algorithm into the rover object (the one mentioned in the questioned)
  + After a rover has been instantiated, there is still scope to change the Rover’s movement algorithm on the fly.

