**Automation test**

PROBLEM : MARS ROVERS

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 on-board cameras can get a complete view of the surrounding terrain to send back to Earth.

A rover's position and location is represented by a combination of x and y co-ordinates and a letter representing one of the four cardinal compass points. The plateau is divided up into a grid to simplify navigation. An example position might be 0, 0, N, which means the rover is in the bottom left corner and facing North.

In order to control a rover, NASA sends a simple string of letters. The possible letters are 'L', 'R' and 'M'. 'L' and 'R' makes the rover spin 90 degrees left or right respectively, without moving from its current spot. 'M' means move forward one grid point, and maintain the same heading.

Assume that the square directly North from (x, y) is (x, y+1).

INPUT:

The first line of input is the upper-right coordinates of the plateau, the lower-left coordinates are assumed to be 0,0.

The rest of the input is information pertaining to the rovers that have been deployed. Each rover has two lines of input. The first line gives the rover's position, and the second line is a series of instructions telling the rover how to explore the plateau.

The position is made up of two integers and a letter separated by spaces, corresponding to the x and y co-ordinates and the rover's orientation.

Each rover will be finished sequentially, which means that the second rover won't start to move until the first one has finished moving.

OUTPUT

The output for each rover should be its final co-ordinates and heading.

INPUT AND OUTPUT

Test Input:

5 5

1 2 N

LMLMLMLMM

3 3 E

MMRMMRMRRM

Expected Output:

1 3 N

5 1 E

Solution:

Rover Class:

**public** **class** Rover {

**public** **static** **final** **char** ***N*** = 'N';

**public** **static** **final** **char** ***E*** = 'E';

**public** **static** **final** **char** ***S*** = 'S';

**public** **static** **final** **char** ***W*** = 'W';

**int** x;

**int** y;

**char** facing;

**public** **void** setPosition(**int** x, **int** y, **char** facing) {

**this**.x = x;

**this**.y = y;

**this**.facing = facing;

}

**public** **void** process(String commands) {

**for** (**char** command : commands.toCharArray()) {

**if** (command == 'M') {

move();

} **else** **if** (command == 'L') {

turnLeft();

} **else** **if** (command == 'R') {

turnRight();

}

}

}

**private** **void** move() {

**switch** (facing) {

**case** ***N***:

y++;

**break**;

**case** ***E***:

x++;

**break**;

**case** ***S***:

y--;

**break**;

**case** ***W***:

x--;

**break**;

}

}

**private** **void** turnLeft() {

**switch** (facing) {

**case** ***N***:

facing = ***W***;

**break**;

**case** ***E***:

facing = ***N***;

**break**;

**case** ***S***:

facing = ***E***;

**break**;

**case** ***W***:

facing = ***S***;

**break**;

}

}

**private** **void** turnRight() {

**switch** (facing) {

**case** ***N***:

facing = ***E***;

**break**;

**case** ***E***:

facing = ***S***;

**break**;

**case** ***S***:

facing = ***W***;

**break**;

**case** ***W***:

facing = ***N***;

**break**;

}

}

**public** String printPosition() {

**return** x + " " + y + " " + facing;

}

}

Rover Test Class:

public class RoverTest {  
  
    private Rover rover;  
  
    @BeforeEach  
    public void setup() {  
        rover = new Rover();  
    }  
  
    @Test  
    public void testRoverInitialization() {  
        rover.setPosition(1, 2, Rover.N);  
        assertEquals(1, rover.x);  
        assertEquals(2, rover.y);  
        assertEquals(Rover.N, rover.facing);  
    }  
  
    @Test  
    public void testRoverMovementNorth() {  
        rover.setPosition(1, 2, Rover.N);  
        rover.process("M");  
        assertEquals(3, rover.y);  
    }  
  
    @Test  
    public void testRoverMovementEast() {  
        rover.setPosition(1, 2, Rover.E);  
        rover.process("M");  
        assertEquals(2, rover.x);  
    }  
  
    @Test  
    public void testRoverMovementSouth() {  
        rover.setPosition(1, 2, Rover.S);  
        rover.process("M");  
        assertEquals(1, rover.y);  
    }  
  
    @Test  
    public void testRoverMovementWest() {  
        rover.setPosition(1, 2, Rover.W);  
        rover.process("M");  
        assertEquals(0, rover.x);  
    }  
  
    @Test  
    public void testRoverTurningLeft() {  
        rover.setPosition(1, 2, Rover.N);  
        rover.process("L");  
        assertEquals(Rover.W, rover.facing);  
    }  
  
    @Test  
    public void testRoverTurningRight() {  
        rover.setPosition(1, 2, Rover.N);  
        rover.process("R");  
        assertEquals(Rover.E, rover.facing);  
    }  
  
    @Test  
    public void testScenarioInput() {  
        rover.setPosition(1, 2, Rover.N);  
        rover.process("LMLMLMLMM");  
        assertEquals("1 3 N", rover.printPosition());  
    }

@Test  
    public void testScenarioForAnotherInput() {  
        rover.setPosition(1, 2, Rover.N);  
        rover.process("MMRMMRMRRM");  
        assertEquals("3 3 E", rover.printPosition());  
    }  
  
}

SOLUTION: MARS ROVERS

Our solution is pretty simple if we look at the x and y graph below, given a scenario that we deployed a robot at position X,Y (0,0) North, I noted the following interesting states and our final solution will be revolving around these rules.

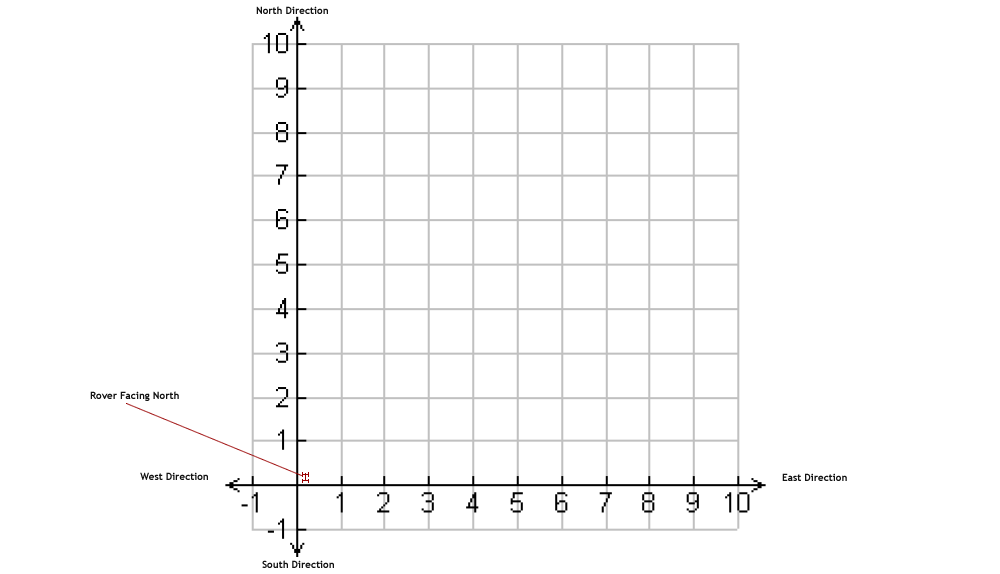
1) Any move from point (X,Y) and the rover facing towards North will increment ++Y

2) Any move from point (X,Y) and the rover facing towards South will decrement --Y

3) Any move from point (X,Y) and the rover facing towards East will increment ++X

4) Any move from point (X,Y) and the rover facing towards West will decrement --X

Finally any change in state of the rover in terms of direction will not affect the positioning of the rover in terms of X,Y location, so let’s say we pass a command string "MM" for a rover positioned at X,Y(0,0) facing North. The final destination of the Rover will be X,Y(0,2) and any state change in direction with command "L" or "R" will only cause the rover to face a new direction in this case West and East respectively but position according to X and Y will remain X,Y(0,2). So with our task clearly defined, Let’s look at the design.



## Solution in Java:

package marsrover;  
public class Rover {  
 public static final Integer N = 1;  
 public static final Integer E = 2;  
 public static final Integer S = 3;  
 public static final Integer W = 4;  
 Integer x = 0;  
 Integer y = 0;  
 Integer facing = N;  
 public Rover() {  
 }  
 public void setPosition(Integer x, Integer y, Integer facing) {  
 this.x = x;  
 this.y = y;  
 this.facing = facing;  
 }  
 public void printPosition() {  
 char dir = 'N';  
 if (facing == 1) {  
 dir = 'N';  
 } else if (facing == 2) {  
 dir = 'E';  
 } else if (facing == 3) {  
 dir = 'S';  
 } else if (facing == 4) {  
 dir = 'W';  
 }  
 System.out.println(x " " y " " dir);  
 }  
 public void process(String commands) {  
 for (int idx = 0; idx < commands.length(); idx ) {  
 process(commands.charAt(idx));  
 }  
 }  
 private void process(Character command) {  
 if (command.equals('L')) {  
 turnLeft();  
 } else if (command.equals('R')) {  
 turnRight();  
 } else if (command.equals('M')) {  
 move();  
 } else {  
 throw new IllegalArgumentException(  
 "Speak in Mars language, please!");  
 }  
 }  
 private void move() {  
 if (facing == N) {  
 this.y ;  
 } else if (facing == E) {  
 this.x ;  
 } else if (facing == S) {  
 this.y--;  
 } else if (facing == W) {  
 this.x--;  
 }  
 }  
 private void turnLeft() {  
 facing = (facing - 1) < N ? W : facing - 1;  
 }  
 private void turnRight() {  
 facing = (facing 1) > W ? N : facing 1;  
 }  
 public static void main(String args[]) {  
 Rover rover = new Rover();  
 rover.setPosition(1, 2, N);  
 rover.process("LMLMLMLMM");  
 rover.printPosition(); // prints 1 3 N  
 rover.setPosition(3, 3, E);  
 rover.process("MMRMMRMRRM");  
 rover.printPosition(); // prints 5 1 E  
 }  
}

**Testing Tasks**

* 1. Write a test for Rover direction facing and positioning initialisation
  2. Write A test for moving the rover x and y position and location
  3. Write a test to move the rover as per the given scenario problem, given input :   
     5 5   
     1 2 N   
     LMLMLMLMM

**Instructions**

There is no right or wrong answer however we are keen to look at how you approach automation with BDD.

Please create a private repo containing your solution on [github.com](https://github.com) and add the following account as a collaborator via settings | manage access: abstract-lgtm