**Introduction**

The solution of the Robot (Rover) problem was achieved by taking a test-driven approach with a focus on unit testing. The chosen programming language is Java and the tests were created and conducted using the unit testing framework for Java called JUnit.

An explanation of the solution is presented covering the analysis, design and implementation.

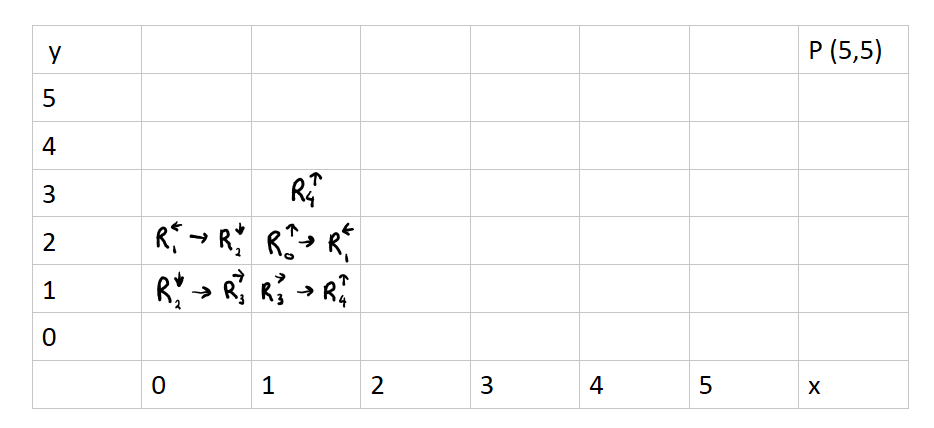
**Analysis**

The approach to the solution was initiated by gaining the understanding of the requirement specification. This was acquired by solving the problem for the example data.

Example 1

Rover’s position: 1 2 N

Instructions: LMLMLMLMM

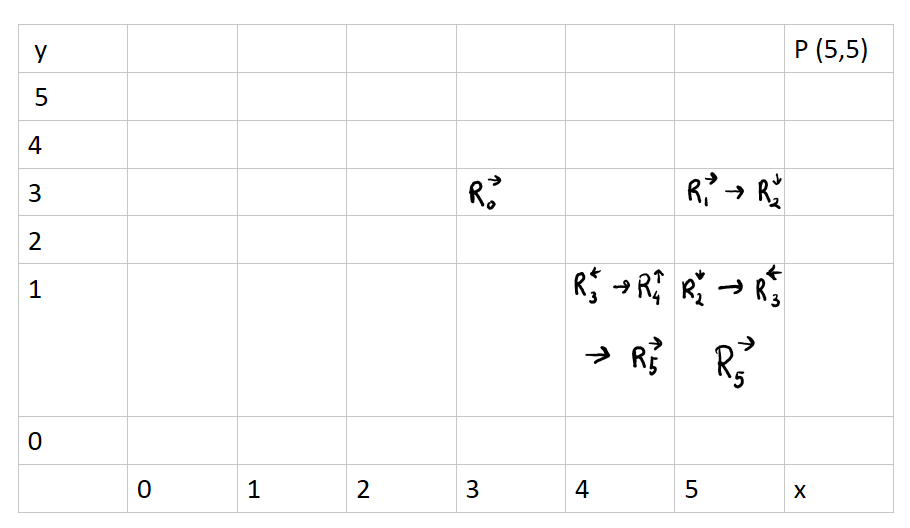


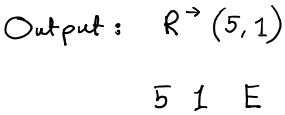
C:\Users\Anum\AppData\Local\Temp\msohtmlclip1\02\clip_image001.png

Example 2

Rover’s position: 3 3 E

Instructions: MMRMMRMRRM





In both of the solutions the Rover’s final position was identical to the expected data.

Following the analysis, it was clear that a Rover entity was essential in the design of the solution. The entity will need specific attributes such as x-position, y-position and the direction of the rover. The rover will require methods to move left or right and move forward. Additionally, a plateau class will be required to create the grid within which the Rover’s movement is permitted.

**Design and Implementation**

As the aim of the assignment was to implement the solution using TDD, the classes of the solution were derived from the tests. Each test was written to meet the requirements and the attributes and methods were created only to fulfil the tests.

The tests were implemented using the “Red, Green, Refactor”:

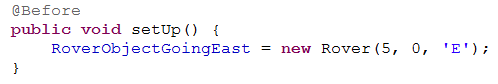
* Red: Create a test and make it fail.
* Green: Make the test pass by any means necessary.
* Refactor: Change the code to remove duplication in your project and to improve the design while ensuring that all tests still pass

The Red/Green/Refactor cycle was iterated rapidly for each new unit of code. The following test shows the progress of the implementation in detail:

Test the movement of the Rover

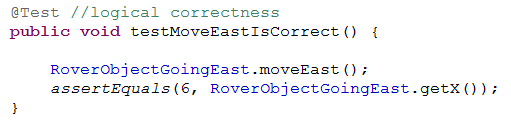
This is one of the tests in the file movementTest.java. The test checks that when a rover is ordered to move to right (towards east) in the grid, the response is as expected.

1. Created the test fixture for a rover going east.



1. To test the logical correctness of a rover going east, a test was created to check that once a rover is asked to move, the rover moves one and only step to the right in the plateau grid.

Created the method moveEast() for the test. The method doesn’t exist in the Rover class yet. The rover’s x-coordinate is expected to be changed from 5 to 6 and it is checked by the function *assertEquals.*



1. Create the method moveEast() in the Rover class. The method is empty and thus, the test will fail.

Machine generated alternative text:
public void moveEast() {
}

1. Ran the test, it failed.

Machine generated alternative text:
Runs: 9/9 0 Errors: O B Failures: 1
ij coordinateTest [Piri ‘ t -.1
J movementTest [Runner JUnit 41(0.000 s)
] testMoveWestlsValid (0.000 s)
testMoveWestlsCorrect (0.000 s)
testMoveEastlsCorrect (0.000 s)

1. Added the logic in the method.

Machine generated alternative text:
public void moveEast() {
mt x = getXQ;
setX(x+1);

1. Rand the test: passed

Machine generated alternative text:
Runs: 9/9 0 Errors: O O Failures: O
-z
coordinateTest [Runner JUnit4] (0.299 s)
movementTest [Runner: JUnit 4) (0.037 s)

The process above was repeated and for each test, a related function was created or an existing function (with different conditions) was used to serve the needs of the system. The tests were derived from the requirement specification and thus, only the functions necessary were created.

The final solution consisted of 6 files – 3 test files, 2 class files and lastly main. The tests are created using fixtures, where the test objects are invoking different methods in various conditions and asserting different expected result. The tests are isolated and each runs a function to create the instance variables for the required objects before running the tests. After each test, a clean-up is issued using the function teardown in the test files.

Description of the source files:

1. coordinateTest.java

The test file consists of the unit tests for the validation of the rover’s coordinates within the specified plateau grid.

Through the tests, the Rover class was implemented containing the x-coordinates, y-coordinates and the direction as the attributes. Another class, Plateau, was implemented to store the details for the grid (x- and y-coordinates). Apart from the getters and setters, a numerous methods were derived from the tests such as:

* checkXCoordinateIsValid()
* checkYCoordinateIsValid()
* checkDirectionIsValid()
* validateRover()

More explanation about the methods can be found in the Rover class.

List of the tests:

* 1. testRoverIsNotNull
  2. testPlateauIsNotNull
  3. testXCoordinateIsValid
  4. testXCoordinateIsInvalid
  5. testYCoordinateIsValid
  6. testYCoordinateIsInvalid
  7. testDirectionIsValid
  8. testDirectionIsInvalid
  9. testRoverIsValid
  10. testRoverIsInvalid

The tests are explained in the comments in the file coordinateTest.java

1. movementTest.java

The test file consists of the unit tests to validate the movement of rover within the plateau grid. Most of the tests validate the rover’s move against the limit of the plateau grid.

List of the tests:

1. testMoveWestIsCorrect
2. testMoveWestIsInvalid
3. testMoveEastIsCorrect
4. testMoveEastIsInvalid
5. testMoveXIsValid
6. testMoveXIsInvalid
7. testMoveNorthIsCorrect
8. testMoveNorthIsInvalid
9. testMoveSouthIsCorrect
10. testMoveSouthIsInvalid
11. testMoveYIsValid
12. testMoveYIsInvalid

The tests are explained in the comments in the file movementTest.java

Methods created from the tests:

* moveWest
* checkMoveXIsValid
* moveEast
* moveNorth
* checkMoveYIsValid
* moveSouth

More explanation about the methods can be found in the Rover class.

1. instructionsTest.java

This test consists of the unit tests to check that Rover is correctly responding to the instructions to turn left or right.

Test list:

* 1. testRoverResponseToLeftIsCorrect
  2. testRoverResponseToRightIsCorrect

The tests are explained in the comments in the file instructionsTest.java

Methods created from the tests:

* turnRight
* turnLeft

1. Rover.java

The class file consists of the attributes/methods to create rover and manipulate its data and location. The data members are accessed through getter and setters while other methods are used to validate the coordinates and movements followed by instructions.

1. Plateau.java

The class file consists of the attributes/methods to define the plateau grid. The file also consist of a function (accessed from main) that retrieves the user input to create the user-defined plateau grid. The coordinates for this grid define the limits within which the rover can move.

1. RoverPlan.java

Contains the main and controls the flow of the system which goes as follows:

1. The system prompts for the coordinates of the plateau grid:
2. The user enters the coordinates (the system checks to ensure that the correct input – integer – has been entered, if not correct - the user will be prompted to enter the coordinates again)
3. The system the grid coordinates which will be used to ensure the rover’s position is within the boundaries.
4. The user is prompted to enter the rover data (the information for a rover’s current position)
5. The user enters the rover’s position (the systems only accepts the specified format of input and will prompt for a new input if the format is not correct)

If the user has entered the rover’s position in the correct format, but the information is not valid considering the rover direction or the location on the grid the user is then prompted to enter valid data for the rover.

1. Once the rover has been validated, the system prompts the for instruction input.
2. The user enters the instructions (only capital letters will be valid). The instructions are iterated, letter by letter, where invalid letters are ignored and the rover follows the instructions till final position is reached.

If the instructions lead to an invalid move outside the plateau grid, the system will notify the user of invalid move and the program will end.

1. The system reveals the final position of the rover.

The program will end.