MANUAL FOR PROBLEM 3

Introduction

Problem 3 for the ThoughtWorks coding test involves a squad of robotic rovers that have been sent to explore a rectangular plateau on Mars. These rovers use a simple coordinate system made up of an (x,y) grid and the cardinal direction the rover is facing: N, S, E, W. The rovers are sent commands from NASA telling the rovers to rotate left or right 90 degrees or move forward one grid point. The objective of the code is to write an object oriented algorithm that maneuvers the rovers in sequence across the Martian plateau and then print out the (x,y) position and the current cardinal direction in which each rover is facing.

Solution

The solution to the problem relies on the object oriented principle of containers. Based on the problem statement, it is clear that NASA will want to store information about both the rectangular plateau on Mars and their fleet of rovers all in the same class called MarsPlateau at NASA headquarters. During the initialization of this class the fleet of rovers will be sent their initial positions and cardinal directions, and their move commands. The rovers, represented by an ArrayList of Rover classes, store the initial position as x and y coordinates, and the move commands as a string. Additionally, the rovers use a Compass and Direction2d class to convert the cardinal directions into a 2 dimensional array representing the current direction each rover will move. This 2 dimensional array is stored in the Direction2d class.

At NASA's command, the MarsPlateau class will be directed to move its fleet of rovers. Each rover goes through their string of move commands one character at a time. The R character causes the rover to rotate right 90 degrees by using the rotate command in Direction2d. The L character causes the rover to rotate left 90 degrees by using the rotate command in Direction2d. Note: the rotate command in Direction2d updates the array stored in that class. The M character causes the rover to move

forward one grid point. This move is done by adding the array stored in Direction2d to the current position of the rover.

After the rovers have been moved, NASA will print out the fleet of rovers current positions and cardinal directions using the print command in the MarsPlateau class. Since the rovers current position is stored in the Rover class, this information can be directly outputted to the command prompt. The cardinal directions on the other hand are retrieved using method calls and then outputted to the command prompt. These method calls are done in two steps. First, the get heading method is used to retrieve the heading from Direction2d. Then the heading is passed through a method in the Compass class which retrieves the cardinal direction. The interplay between the Compass class and Direction2d will be discussed in a later section of this manual.

Compilation

To compile the Java source code, open up a terminal and go to the directory problem_3 using terminal commands. The exact command depends on the OS. Once the terminal is in the problem_3 directory, type “javac *.java” to compile the source code into .class files. Do not close the terminal at this point. It will be used to execute the solution to problem 3 in the next step.

Executing Using Main Method

There are two ways to execute the solution to problem 3. The first is to execute the main method in the problem_3 package and the second is to import the problem_3 package and use methods from the MarsPlateau class. In this section, the first method of execution will be explained.

To execute the first method, you will need to move to the parent directory of problem_3. In the parent directory type “java problem_3/Main” and then the name of the text file where the input for the problem is stored. Press enter and the program will load the specified text file and output the solution to the terminal.

The text file has a very specific format. The first line of the text file will be xmax and ymax of the MarsPlateau which corresponds to the upper-right coordinates. The

lower-left coordinates do not need to be specified since they are assumed to be 0,0. The rest of the lines of input are information required for each Rover to operate. Each Rover is defined by two lines. The first line is x_position y_position cardinal_direction. The second line is the move_commands. An example of an input file can be found below:

5 5

1 2 N

LMLMLMLMM

3 3 E

MMRMMRMRRM

Importing the Package and Executing

Importing the problem_3 package into your source code requires opening the source file and adding “import problem_3.*; ” directly below the package statement if one exists. If your goal is to solve problem 3 continue reading. Otherwise, skip to the section on reusing generic classes to learn how to use the Compass and Direction2d class for your project needs.

To solve problem_3, you will want to use the commands below.

```
MarsPlateau mars = new MarsPlateau(test);

mars.move_rovers();

mars.print_rovers();
```

The variable test is a string which contains text in the format described in the section above. This string can be built using a text file or by assignment.

Add the commands above and your method of choice for building the string test into the correct format to your source code. Recompile your source file and rerun. The code should now be able to solve problem 3.

Error Handling

There are six common errors that can occur while executing the solution to problem 3. In this section, the error handling for these five situations is explained in detail.

When running problem 3 from the main method, the code will load a file. If this file does not exist, an exception occurs returning a FileNotFoundException. If the read in file or inputted string for initializing the MarsPlateau class does not match the correct format, an exception occurs. This exception is caught and an explanation of the correct format is outputted to the terminal. Additionally, if any rovers are constructed, they will continue to function normally.

Due to possible errors from space communication and the fleet of rovers operating on Mar's surface, the final four errors are not handled with exceptions. The first of these errors is the rover moving off the assigned grid. Because space communication is fairly slow, this error is not reported to NASA. Instead, the rover is forcefully moved back onto the assigned grid. Another common error is an invalid move command. In this case, the rover ignores the command and moves to the next command in the sequence. This error is also not reported back to NASA for the same reason as the previous error. The final two errors, invalid heading and invalid compass point are reported back to NASA. These errors can occur when trying to convert a heading to a cardinal direction or a cardinal direction to a heading. If the heading cannot be converted to a cardinal direction, a serious problem may have occurred with the rovers internal navigation system. This error is reported to NASA with the following text "The heading cannot be converted to a compass point." The other conversion error is less serious and the error is reported to NASA as "Invalid compass point Returning default heading of 0 degrees".

Reusing Generic Classes

There are two generic classes developed in this problem: Compass and Direction2d. Generic in this case means these classes can be reused for other problems. The doc strings for these classes and their public methods are shown below.

```
/**
 * Generic class which stores the compass rose and allows conversion
between compass points and headings in degrees.
 * The headings in this class are done in such a way that east is 90
degrees and N is 0 degrees.
 * This is the conventional way of representing compass headings.
 * @author zacharykraus
 *
 */
public class Compass {

/**
   * Constructor which builds the compass rose stored in this class.
   * The compass rose is a HashMap with keys that are the compass
points
   * and values that are the headings in degrees.
   */
  public Compass()

  /**
   * Method uses the compass point to get the heading stored in
compass rose
   * @param input the compass point
   * @return The heading in degrees. Currently N is 0 degrees and E is
90 degrees.
   */
  public double getHeading(String input)

  /**
   * Method uses the heading to get the compass point stored in
compass rose
   * @param angle corresponding to the heading
   * @return The compass point
   */
  public String getCompassPoints(double angle)
}

/**
 * Generic class which stores the direction in 2 dimensions an object
will move each time step.
 * The class allows the positions of objects to be updated each time
step and
 * the direction to be rotated but not increased in magnitude.
 * Additionally, the class allows calculating both the heading and the
angle.
 * @author zacharykraus
 *
 */
public class Direction2d {
```

```
/**
  * Constructor which uses an array to build the class
  * @param input array used to construct the class
  */
  public Direction2d(double [] input)

/**
   * Constructor which uses an angle and the distance to build the
class
   * @param angle in degrees where 90 degrees corresponds to north or
the vector (1,0)
   *  and 0 degrees corresponds to east or the vector (0,1)
   * @param distance the object moves per time step
   */
  public Direction2d(double angle, double distance)

  /**
   * Constructor which uses a compass, a compass_point and the
distance to build the class
   * @param compass class which contains a map relating compass points
to headings
   * @param compass_point a string that is on the compass rose ie: E,
W, SE, etc., etc.
   * @param distance the object moves per time step
   */
  public Direction2d(Compass compass, String compass_point, double
distance)

/**
   * Method gets the direction array stored in the class
   * @return the direction array
   */
  public double [] getDirection()

/**
   * Method gets the x value in the direction array stored in the
class
   * @return the x value in the direction array
   */
  public double x()

  /**
   * Method gets the y value in the direction array stored in the
class
   * @return the y value in the direction array
   */
  public double y()
```

```
/**
   * Method takes the current direction an object is moving in and
calculates the angle
   * @return the angle in degrees where 0 degrees corresponds with the
vector (1,0) and 90 degrees with (0,1).
   * If x and y in the direction array are both 0, this method returns
NaN.
   */
  public double angle()

  /**
   * Method takes the current direction an object is moving in and
calculates the heading
   * @return the heading in degrees where 0 degrees corresponds with
the vector (0,1) and 90 degrees with (1,0).
   * If x and y in the direction array are both 0, this method returns
NaN.
   */
  public double heading()

/**
   * Method rotates the direction array stored in the class by the
angle inputed
   * @param angle in degrees where positive rotates counter clockwise
and
   * negative rotates clockwise
   */
  public void rotate(double angle)
}
```

Note: if you want to change the distance the object moves per time step, you need to construct a new Direction3d object with the new distance value. In future updates, new methods may be added to allow the distance to be changed without having to construct a new object.