G51OOP Coursework 4 –Mars Rover

Question

For this coursework you are to create a Mars Rover simulator. Several classes are provided, you should load these into an Eclipse project. Some classes are completed for you, others have a varying degree of information in them. This coursework probably requires less coding than previous work, but is conceptually more difficult. You should read the whole specification first, and consider drawing how the classes will interact with each other before beginning to write your code.

Device Driver (Provided as .class file)

The DeviceDriver class represents the connection to the actual hardware. In the real world this class would be written in C and assembly and talk directly to the hardware, however for this exercise it is provided as a java class. Javadoc for the DeviceDriver class can be found at:

http://www.cs.nott.ac.uk/~cah/javadocDeviceDriver/index.html

Note each of the methods may throw HardwareFailExceptions, these should be caught and handled appropriately.

TestMain

Your TestMain class should contain the main method. This class should be responsible for parsing commands from user input (you may use G51OOPInput) and creating the appropriate type of Command object. Note there are no menus in this coursework, only a command line prompt which should be "> " (without the quotes!). These Commands are then to be sent to your Rover class via the receiveCommand(Command command) method. Parsing should allow for both uppercase and lowercase characters, and should be parsed using the rules below:

- A valid command from standard input has one or two parts (depending on whether a particular Command has an argument), separated by a single space.
- The format is: command <argument> where command can be one of:
    - 'T' or 't' – This is a get temperature command. The argument should be 'C' or 'F', representing Celcius or Fahrenheit. eg "T C"
    - 'P' or 'p' – This is a get pressure command. It has no argument.
    - 'M' or 'm' – This is a drive comment, it has one argument representing the number of spaces. It may be positive or negative. eg "M -5"
    - 'R' or 'r' – This is a turn (rotate) command. It is followed by an argument L (or l) for left, R (or r) for right, or A (or a) for around. eg "R L"

    - ‘L’ or ‘l’ – This is a get location command. It has no argument. Eg “L”
    - ‘Q’ or ‘q’ – This is a quit command and should cause the program. To terminate. It has no argument. Eg “q”
    - There is no command object for getting the mission time, the mission time is included in every response. (See Response)
- If a command is not valid then an “ErrorResponse” should be returned. Your main should be able to handle both uppercase and lowercase commands. A command without the correct argument (e.g move without an integer) is invalid.
- Failure to parse commands correctly will result in a large loss of marks. In the real world it is likely that commands would be read from another program rather than typed by hand, so accuracy is important. If you have questions about the commands, please ask on the Moodle forum.

When a response is returned to the testMain object, it should be printed by using the toString method on the response object. This output should match the expected output. (see expected output documentation for details).

Rover

The Rover should have several Assets. These Assets are GPSAsset (for location), SteererAsset, ThermometerAsset, BarometerAsset, DriveAsset. When a Command is received via the receiveCommand() method, the rover should dispatch the Command object to the correct Asset for handling. eg, a getTemperature() command should go to the ThermometerAsset. The Asset will return a Response, which the Rover should add to a log (see Response below). The Rover should also have a method getLog() which returns an ArrayList of all Responses so far. Finally, the Rover should return a Response from the receiveCommand() method (which the Main object will print out to standard output).

Asset

Asset is an abstract class that provides some common functionality to its subclasses. You should carefully consider what should go in this class, as certain functionality is shared between different types of asset. Each Asset is responsible for calling its appropriate DeviceDriver method.

Commands

Command is an abstract class which provides a toString() method. Commands inherited from this class should hold all of the information required for that specific Command to be carried out. There is a different type of Command for each sensor.

Responses

Response is an abstract class which your response types should use. A Response should have a toString() method to provide output as per the expected output documentation. The response should include the start time, end time, result **and original command object**. The result depends on the type of command, it may be a direction, integer, double, Location or String (for an ErrorRersponse).

Error Conditions

- Exceptions thrown from the DeviceDriver should be caught and added to the response. If an exception is caught, an ErrorResponse should be created containing the message provided in the Exception (use the getMessage() method). If for any reason the Rover has lost signal, you will not be able to retrieve the mission time or location. These should have a value of 0 (zero).
- Invalid commands should return an ErrorResponse including the message "Invalid command". The startTime and endTime can be retrieved using the DeviceDriver.getMissionTime() method, if this is not possible (No signal), these should have a value of 0 (zero).

Hints

- You will have many different types of Command, and many different types of Response. You will need to store these responses in a log in the Rover class. You will also have many different types of Asset. You should consider the use of inheritance for these Commands, Response and Assets.
- Exceptions should be caught and turned into an error response. The Assets are responsible for creating the Response objects (unless it is an invalid command). You should consider where to catch these exceptions. Note, no exception should cause the program to exit.

Notes

All output, including error messages, should be printed on standard output (not the error stream).

You have one (1) submission. Requests for re-submission with reason stated need to be addressed to Colin Higgins. If anything goes wrong with a submission, including submitting the incorrect files, then the time of the re-submission will be applied to this coursework, if the re-submission is allowed.

You can be up to two days late with standard University penalties applied (ie 5% per day or part thereof). Later than 2 days will attract a 100% penalty (ie no marks will be awarded).

You may have problems getting the DeviceDriver to work as we are only providing the.class file. If so try the following steps:

1 - Create new Java project in eclipse
2 - Drag all of the .java files into the src folder.
3 - Right click the project, go to Build Path -> Configure Build Path
4 - Click the "Libraries" tab.
5 - Click "Add External Class Folder"

6 - Find the folder with the .class in, select it and press OK.

Then you can use DeviceDriver.getTemperature etc properly. It will add that class to the build path so it is included by eclipse when compiling.

Typical Input/Output

Normal usage

```
>  T C
startTime = 0; endTime = 1; command = GetTemperatureCommand C;
temperature is -25.607017003965517
>  T F
startTime = 1; endTime = 2; command = GetTemperatureCommand F;
temperature is -14.092630607137934
>  R L
startTime = 2; endTime = 202; command = MakeRotateCommand LEFT;
direction is WEST
>  M 2
startTime = 202; endTime = 302; command = MakeMoveCommand 2; distance is
2
>  P
startTime = 302; endTime = 304; command = GetPressureCommand; pressure is
7402
>  T C
startTime = 304; endTime = 305; command = GetTemperatureCommand C;
temperature is -30.91168824543142
>  L
startTime = 305; endTime = 305; command = GetLocationCommand; location is
[-2, 0] WEST
>  Q
Quitting
```

Invalid Commands

```
>  T T
TestMain::generateCommand, parse error: incorrect temperature value T
> M M
TestMain::generateCommand, parse error: incorrect move value M
>   R B
TestMain::generateCommand, parse error: incorrect rotation type B
```

```
> e
TestMain::generateCommand, parse error: incorrect command option
> P L
TestMain::generateCommand, parse error: incorrect command length
> L 234df
TestMain::generateCommand, parse error: incorrect command length
> Q
Quitting
```

Stuck in the mud (HardwareFailException handling)

```
> M 7
startTime = 0; endTime = 100; command = MakeMoveCommand 7; distance is 6
> M 1
startTime = 0; endTime = 0; command = MakeMoveCommand 1; error is: The
wheels have become stuck, try turning left/right to free the rover.
> R L
startTime = 0; endTime = 0; command = MakeRotateCommand LEFT; error is:
The rover is stuck, but the wheels moved a little, keep trying..
> R R
startTime = 0; endTime = 0; command = MakeRotateCommand RIGHT; error is:
The rover is stuck, but the wheels moved a little, keep trying..
> R L
startTime = 0; endTime = 0; command = MakeRotateCommand LEFT; error is:
The rover is stuck, but the wheels moved a little, keep trying..
> R R
startTime = 0; endTime = 0; command = MakeRotateCommand RIGHT; error is:
The rover is stuck, but the wheels moved a little, keep trying..
> R L
startTime = 100; endTime = 300; command = MakeRotateCommand LEFT;
direction is WEST
> M 1
startTime = 300; endTime = 400; command = MakeMoveCommand 1; distance is
1
> Q
Quitting
```

Mark Scheme

You will be marked on correctness (if your output is correct for the given input). There are also marks for variable naming, layout, efficiency and output format. Marks will be given for correct use of inheritance, abstract classes, interfaces, array lists and class

responsibility. The format of the output should match the format of the expected output. The complexity and structure of the methods you use will be assessed. As with any programming, you should use comments where necessary in your code to make it more readable. You may be marked down for failure to submit the correct files, or follow the specification (See submission). You will also lose marks for correctness if you have clearly made no attempt to use objects and classes (ie a procedural solution will detract from the correctness marks).

Submission

Submission is via Moodle. You should submit a .zip file containing all required .java files needed to compile the work. The class containing the main method should be named TestMain.java. Your work should compile using the command "javac TestMain.java" and run using "java TestMain". It should run on the machines available in the lab. You should not submit any other format than .zip (no .iso, .rar, .7z or other alternatives). Make sure to submit the .java files, and not the .class files!