Please provide your best knowledge of Object Oriented Programming when solving exercises where it makes sense. It is expected that you will provide unit tests to show that the code works.

Exercise #1:

A robot lands on Mars, which happens to be a cartesian grid; assuming that we hand the robot these instructions, such as LFFFRFFFRRFFF, where "L" is a "turn 90 degrees left", "R" is a "turn 90 degrees right", and "F" is "go forward one space, please write control code for the robot such that it ends up at the appropriate-and-correct destination, and includes unit tests.
Here is an example output with command "FF":
[0, 2]

Exercise #2:

Part 1:
Write a program that counts in sequential order when given a start and end value - without using any iterative programing loops, i.e. while, for, do, for-each, etc. You can assume that both the start and end values will always be positive and that the start value will always be less then the end value. There should only be one method with the following signature:

```
void countUp(int start, int end) {
// All code exercise code should go here
}
```

Here is example output with start=0 and end=5:

```
0
1
2
3
4
5
```

Part 2:
Continuing with part 1 change the output of the test, so that it now prints out in sequential order to the end value (only once), but then also counts down to the start value. Again, using no iterative loops, and assuming that both the start and end values will always be positive and that start value will always be less then the end value. There should only be one method with the following signature:

```
void countUpAndDown(int start, int end) {
// All code exercise code should go here
}
```

Here is example output with start=0 and end=5:

```
0
1
2
3
4
5
4
3
2
1
0
```

Exercise #3:

Using the robot code/idea, create a “robot race”, of sorts:

- This race will consist of several different robots (robots like competition)
- Each robot will start the race at coordinate [0,0] (x,y), facing North/Up
- Each robot will have a name. It doesn’t have to be unique, although...it will be helpful for viewing purposes!
- Each robot will iterate/process a series of actions: one of “L”, “R”, or “F”, just as in the previous exercise.
- Each robot will have a delay between executing actions. Each robot can have its own delay. The delay for a robot is decided at robot creation time, and remains the same for the life of the robot.
- Each robot will have the same starting coordinates.
- These robots are “ghosts”, in that they can pass through one another, occupy the same space at the same time, without any issues.
- Each robot will be given its string of actions at the beginning of the race. When you say “Go!” (start the program), each robot will simultaneously start processing through their own actions, according to the delay they have been assigned. For each action they complete, they should tell “the audience” (print to System.out) what they have done.
- Each robot will also state the elapsed number of seconds when announcing the action most recently performed (this does not have to be perfect; hint: Thread.sleep(...) or similar would be acceptable).
- A robot’s personal “finish line” is simply the end of its action command set. There is no need to navigate robots to a common end coordinate.
- When a robot reaches the end of its journey, it should receive (and remember) a rank. The first to stop will get 1, the second will get 2, and so on. It also should know what coordinate it has ended on, relative to its starting point.
- Once ALL robots have completed the race (executed all of their actions), each one will tell “the audience” (print to System.out) what its rank and final coordinate position is. Each robot will wait its turn (so the winner prints first, followed by second place, etc).
- Once all robots have exclaimed their ranking and coordinate position, the program should terminate.

For example, if the following is true:

Joe has a delay of 3 seconds between actions, and has actions: FRFLLFRF
Bill has a delay of 1 second between actions, and has actions: FFFFFLF
Jim has a delay of 2 seconds between actions, and has actions: LFRF

This could be the program’s output:

```
The race has STARTED!
01s  Joe: Moves F
01s  Bill: Moves F
01s  Jim: Turns L
02s  Bill: Moves F
03s  Bill: Moves F
03s  Jim: Moves F
04s  Joe: Turns R
04s  Bill: Moves F
05s  Bill: Moves F
05s  Jim: Turns R
06s  Bill: Turns L
07s  Bill: Moves F
07s  Joe: Moves F
07s  Jim: Moves F
10s  Joe: Turns L
13s  Joe: Turns L
16s  Joe: Moves F
19s  Joe: Turns R
22s  Joe: Moves F
22s  The race has ENDED!
```

```
Bill is #1, at [-1,5]
Jim is #2, at [-1,1]
Joe is #3, at [0,2]
```
[Program Terminates]

Please try to design the program so that it could easily be extended in the future to support things such as (hint: these may be future talking points):

- A “misguided robot” that turns/moves in the opposite direction of its commands
- A “speedy robot” that moves forward 3x as far as it is commanded to
- Battery powered robots, that might stop executing commands if they expended too much energy.