ParkingLot v2.0.0

## Nature of the Game

We want to understand how you think as a programmer, and the level of craft you bring to bear when building software.

Of course, the ideal would be a real world problem, with real scale, but that isn't practical as it would take too much time. So instead we have a dead simple, high school level problem that we want you to solve *as though* it was a real-world problem.

Please note that not following the instructions below will result in an automated rejection. Taking longer than the time allocated will negatively affect our evaluation of your submission.

## Rules of the Game

1. You have two full days to implement a solution.
2. We are really, really interested in your object oriented or functional design skills, so please craft the most beautiful code you can.
3. We're also interested in understanding how you make assumptions when building software. If a particular workflow or boundary condition is not defined in the problem statement below, what you do is your choice.
4. You have to solve the problem in any object oriented or functional language **without using any external libraries** to the core language except for a testing library for TDD. Your solution **must** build+run on Linux. If you don't have access to a Linux dev machine, you can easily set one up using Docker.
5. Please use Git for version control. We expect you to send us a **standard zip or tarball** of your source code when you're done that includes Git metadata (the .git folder) in the tarball so we can look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus.
6. Please **do not** check in binaries, class files, jars, libraries or output from the build process.
7. Please write comprehensive unit tests/specs. For object oriented solutions, it's a huge plus if you test drive your code. Submitting your solution with only a functional suite will result in a rejection.
8. Please create your solution inside the `parking_lot` directory. Your

codebase should have the same level of structure and organization as any mature open source project including coding conventions, directory structure and build approach (make, gradle etc) and a README.md with clear instructions.

9. For your submission to pass the automated tests, please update the Unix executable scripts `bin/setup` and `bin/parking_lot` in the bin directory of the project root. `bin/setup` should install dependencies and/or compile the code and then run your unit test suite. `bin/parking_lot` runs the program itself. It takes an input file as an argument and prints the output on STDOUT. Please see the examples below. Please note that these files are Unix executable files and should run on Unix.

10. Please ensure that you follow the syntax and formatting of both the input and output samples. The zip file you have been sent includes the same automated functional test suite we use. This is to help you validate the correctness of your program. You can run it by invoking `bin/run_functional_tests`. **IMPORTANT:** To make the functional specs work correctly, some setup is needed. Instructions to set up the functional suite can be found under `functional_spec/README.md`.

11. Please do not make either your solution or this problem statement publicly available by, for example, using github or bitbucket or by posting this problem to a blog or forum.

## Problem Statement

I own a parking lot that can hold up to 'n' cars at any given point in time. Each slot is given a number starting at 1 increasing with increasing distance from the entry point in steps of one. I want to create an automated ticketing system that allows my customers to use my parking lot without human intervention.

When a car enters my parking lot, I want to have a ticket issued to the driver. The ticket issuing process includes us documenting the registration number (number plate) and the colour of the car and allocating an available parking slot to the car before actually handing over a ticket to the driver (we assume that our customers are nice enough to always park in the slots allocated to them). The customer should be allocated a parking slot which is nearest to the entry. At the exit the customer returns the ticket with the time the car was parked in the lot, which then marks the slot they were using as being available. Total parking charge should be calculated as per the parking time. Charge applicable is $10 for first 2 hours and $10 for every additional hour.

We interact with the system via a simple set of commands which produce a specific output. Please take a look at the example below, which includes all the commands

you need to support - they're self explanatory. The system should accept a filename as a parameter at the command prompt and read the commands from that file.

**Example: File**

To install all dependencies, compile and run tests:

```
$ bin/setup
```

To run the code so it accepts input from a file:

```
$ bin/parking_lot file_inputs.txt
```

***Commands***

- `Create parking lot of size n : create_parking_lot {capacity}`
- `Park a car : park {car_number}`
- `Remove(Unpark) car from : leave {car_number} {hours}`
- `Print status of parking slot : status`

***Input (contents of file):***

```
create_parking_lot 6
park KA-01-HH-1234
park KA-01-HH-9999
park KA-01-BB-0001
park KA-01-HH-7777
park KA-01-HH-2701
park KA-01-HH-3141
leave KA-01-HH-3141 4
status
park KA-01-P-333
park DL-12-AA-9999
leave KA-01-HH-1234 4
leave KA-01-BB-0001 6
leave DL-12-AA-9999 2
park KA-09-HH-0987
park CA-09-IO-1111
park KA-09-HH-0123
status
```

***Output (to STDOUT):***

```
Created parking lot with 6 slots
Allocated slot number: 1
Allocated slot number: 2
Allocated slot number: 3
Allocated slot number: 4
Allocated slot number: 5
Allocated slot number: 6
```

```
Registration number KA-01-HH-
3141 with Slot Number 6 is free with Charge 30
Slot No.  Registration No.
1   KA-01-HH-1234
2   KA-01-HH-9999
3   KA-01-BB-0001
4   KA-01-HH-7777
5   KA-01-HH-2701
Allocated slot number: 6
Sorry, parking lot is full
Registration number KA-01-HH-
1234 with Slot Number 1 is free with Charge 30
Registration number KA-01-BB-
0001 with Slot Number 3 is free with Charge 50
Registration number DL-12-AA-9999 not found
Allocated slot number: 1
Allocated slot number: 3
Sorry, parking lot is full
Slot No.  Registration No.
1   KA-09-HH-0987
2   KA-01-HH-9999
3   CA-09-IO-1111
4   KA-01-HH-7777
5   KA-01-HH-2701
6   KA-01-P-333
```