Rules of the Game

1. You have two full days to implement a solution.
2. We are really, really interested in your object oriented development skills, so please solve the problem keeping this in mind.
3. 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, a README.md with clear instructions and (additionally) a `parking_lot` shell script that automates the entire build and execute process.
4. MANDATORY: You have to solve the problem in Ruby, Java, Go or any other object oriented language **without using any external libraries/gems** except for a testing library for TDD. Your solution **must** build+run on Linux.
5. MANDATORY: 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. MANDATORY: Please **do not** check in class files, jars or other libraries or output from the build process. Use a standard build automation and dependency system like ant/maven/rake.
7. MANDATORY: Please write comprehensive unit tests/specs. Additionally, it's a huge plus if you test drive your code.
8. MANDATORY: Please ensure that you follow the syntax and formatting of both the input and output samples. We validate submissions using automated tests. For your submission to pass the automated tests, **please include an executable file called parking_lot at the root of your project directory** which builds the code, runs tests/specs, then runs the program. It takes an input file as an argument and prints the output on STDOUT. Please see the example below.
9. MANDATORY: 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.
10. MANDATORY: Please add a README file with relevant details.

Problem Statement

I own a multi-storey 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 which then marks the slot they were using as being available.

Due to government regulation, the system should provide me with the ability to find out:

- Registration numbers of all cars of a particular colour.
- Slot number in which a car with a given registration number is parked.
- Slot numbers of all slots where a car of a particular colour is parked.

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 allow input in two ways. Just to clarify, the same codebase should support both modes of input - we don't want two distinct submissions.

1) It should provide us with an interactive command prompt based shell where commands can be typed in

2) It should accept a filename as a parameter at the command prompt and read the commands from that file

Example: File

To run the program:

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

Input (contents of file):

```
create_parking_lot 6
park KA-01-HH-1234 White
park KA-01-HH-9999 White
park KA-01-BB-0001 Black
park KA-01-HH-7777 Red
park KA-01-HH-2701 Blue
park KA-01-HH-3141 Black
leave 4
status
park KA-01-P-333 White
park DL-12-AA-9999 White
registration_numbers_for_cars_with_colour White
slot_numbers_for_cars_with_colour White
slot_number_for_registration_number KA-01-HH-3141
slot_number_for_registration_number MH-04-AY-1111
```

Output (to STDOUT):

```
Created a 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
Slot number 4 is free
Slot No.  Registration No     Colour
1    KA-01-HH-1234  White
2    KA-01-HH-9999  White
3    KA-01-BB-0001  Black
5    KA-01-HH-2701  Blue
6    KA-01-HH-3141  Black
Allocated slot number: 4
Sorry, parking lot is full
KA-01-HH-1234, KA-01-HH-9999, KA-01-P-333
1, 2, 4
6
Not found
```

Example: Interactive

To run the program and launch the shell:

```
$ ./parking_lot
```

Assuming a parking lot with 6 slots, the following commands should be run in sequence by typing them in at a prompt and should produce output as described below the command:

```
Input:
create_parking_lot 6

Output:
Created a parking lot with 6 slots

Input:
park KA-01-HH-1234 White

Output:
Allocated slot number: 1

Input:
park KA-01-HH-9999 White

Output:
Allocated slot number: 2

Input:
park KA-01-BB-0001 Black

Output:
Allocated slot number: 3

Input:
park KA-01-HH-7777 Red

Output:
Allocated slot number: 4

Input:
park KA-01-HH-2701 Blue

Output:
Allocated slot number: 5

Input:
park KA-01-HH-3141 Black

Output:
Allocated slot number: 6

Input:
leave 4
```

Output:
Slot number 4 is free

Input:
status

Output (tab delimited output):

Slot No	Registration No.	Colour
1	KA-01-HH- 1234	White
2	KA-01-HH- 9999	White
3	KA-01-BB- 0001	Black
5	KA-01-HH- 2701	Blue
6	KA-01-HH- 3141	Black

Input:
park KA-01-P-333 White

Output:
Allocated slot number: 4

Input:
park DL-12-AA-9999 White

Output:
Sorry, parking lot is full

Input:
registration_numbers_for_cars_with_colour White

Output:
KA-01-HH-1234, KA-01-HH-9999, KA-01-P-333

Input:
slot_numbers_for_cars_with_colour White

Output:
1, 2, 4

Input:
slot_number_for_registration_number KA-01-HH-3141

Output:

6

Input:
slot_number_for_registration_number MH-04-AY-1111

Output:
Not found