Task: Engineer: Java: v3

Rules

1. You have five days to implement the 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 `warehouse` shell script that automates the entire build and execute process.
4. **MANDATORY**: You have to solve the problem in Java using Spring (Boot) framework,and with no dependency on any other library 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 maven / gradle.
7. **MANDATORY**: Please write comprehensive unit tests/specs.
8. **MANDATORY**: Please ensure that you follow the syntax and formatting of both the input and output samples. For your submission to pass the automated tests, please include an executable file called `warehouse` 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

- We own a warehouse that can hold upto 'n' products at a time. The warehouse has multiple racks, and each rack is assigned a slot no. A product is always assigned the nearest slot no. For eaxmple, the product that comes first will be assigned the slot 1. We want to create an automated system where in if a product enters the warehouse it can be placed into a slot without any human intervention.

- When a product enters the warehouse, a receipt is generated in the system. The receipt generation process includes documenting the product code (12-digit Unique Number), color and allocating an available slot to the product. The product should be allocated a slot which is nearest to the entry. When a product is sold, based on the receipt of that product, the slot is marked as available.

- The system should allow me to find out
    - Product code of all products of a particular color
    - Slot number in which a product with a given product code is placed
    - Slot numbers of all the slots where a product of a particular color is stored

- 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.

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

Example: System Interaction

To run the program and launch the shell:

```
$ ./warehouse
```

Assuming a warehouse 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:
warehouse 6

Output:
Created a warehouse with 6 slots

Input:
store 72527273070 White

Output:
Allocated slot number: 1

Input:
store 72527273071 Green

Output:
Allocated slot number: 2

Input:
store 72537113170 Purple

Output:
Allocated slot number: 3

Input:
```

```
store 72537113171 Black

Output:
Allocated slot number: 4

Input:
store 72537113172 Black

Output:
Allocated slot number: 5

Input:
store 72537113173 Green

Output:
Allocated slot number: 6

Input:
sell 4

Output:
Slot number 4 is free

Input:
status

Output (tab delimited output):
Slot No. Product Code  Colour
1        72527273070   White
2        72527273071   Green
3        72537113170   Purple
5        72537113172   Black
6        72537113173   Green

Input:
store 82537113174 Purple

Output:
Allocated slot number: 4

Input:
store 82527273075 White

Output:
Warehouse is full

Input:
product_codes_for_products_with_colour  White

Output:
72527273070, 82527273075
```

```
Input:
slot_numbers_for_products_with_colour  Green

Output:
2, 6

Input:
slot_number_for_product_code  82537113174

Output:
4

Input:
slot_number_for_product_code  82532222174

Output:
Not found
```

Bonus (Optional Step)

- Additionally,
    - You can design a set of REST APIs to manage all the above, and
    - Persist all the data in a database of your choice. Ensure the database schemas and tables are automatically created at the start of the system.

Languages / Tools

- Language to be used : Java
    - Preferred Version : (8+)
- Framework : Spring
    - Preferred Version : (4.x +)
    - Preferred : Spring Boot (1.4+)
- Build Tools : Maven or Gradle