Engineering Challenge Homework

Background

At City Storage Systems, we assist in the production and fulfillment of online food orders. In the process of creating and delivering these orders, some common workflows involve processing and displaying large amounts of order events — when orders are placed, prepared, delivered, etc. These orders and events can be displayed on interfaces used by cooks, delivery drivers, in-house engineers, and more. You will be building a (simulated) system to help facilitate this.

Challenge Prompt

Create a mobile Android or iOS application that emulates the fulfillment of delivery orders for a kitchen. The kitchen should receive 2 delivery orders per second. The kitchen should instantly cook the order upon receiving it, and then place the order on the best-available shelf (see *Shelves section*) to await pick up by a courier.

Upon receiving an order, the system should dispatch a courier to pick up and deliver that *specific* order. The courier should arrive *randomly* between 2-6 seconds later. The courier should instantly pick up the order upon arrival. Once an order is picked up, the courier should instantly deliver it.

You can use any programming language and framework to demonstrate your best work, though we tend to most commonly see the Android or iOS platform defaults. Similarly your IDE and build environment are likely dictated by the platform, though you can document any callouts in the readme.

User Interface

Your application should show some form of display that allows interviewers to clearly follow your system's operations **as it runs in real-time**. Whenever events occur in your system (order received, picked up, discarded, etc), your system **should output both** the triggering event and a full listing of shelves' contents.

However **the UI is not the focus of this exercise**. A submission is scored based on the order processing functionality alone (see rubric below). Your application's UI can include any representation you want: a simple text view log, a table/list view, or some visual representation of the shelves. A snazzy UI definitely isn't required for a successful submission, but can be a favorable addition.

Orders

You can download a JSON file of food order structures from **http://bit.ly/css_dto_orders**. Orders must be parsed from the file and ingested into your system at a rate of 2 orders per second. You are expected to make your order ingestion rate configurable, so that we can test your system's behavior

with different ingestion rates. Each order should only be ingested once; when all orders have been consumed, your system should terminate.

orders.json

```
[
   {
      "id": "0ff534a7-a7c4-48ad-b6ec-7632e36af950",
      "name": "Cheese Pizza",
      "temp": "hot",          // Preferred shelf storage temperature
      "shelfLife": 300,       // Shelf wait max duration (seconds)
      "decayRate": 0.45       // Value deterioration modifier
   },
   ...
]
```

Shelves

The kitchen pick-up area has multiple shelves to hold cooked orders at different temperatures. Each order should be placed on a shelf that matches the order's temperature. If that shelf is full, an order can be placed on the overflow shelf. If the overflow shelf is full, an existing order of your choosing on the overflow should be moved to an allowable shelf with room. If no such move is possible, a random order from the overflow shelf should be discarded as waste (and will not be available for a courier pickup). The following table details the kitchen's shelves:

Name	Allowable Temperature(s)	Capacity
Hot shelf	**`Hot`**	10
Cold shelf	**`Cold`**	10
Frozen shelf	**`Frozen`**	10
Overflow shelf	Any temperature	15

Shelf Life

Orders have an inherent value that will deteriorate over time, based on the order's **`shelfLife`** and **`decayRate`** fields. Orders that have reached a value of zero are considered wasted: they should never be delivered and should be removed from the shelf. Please display the current order value when displaying an order in your system's output.

```
Order Value Formula
```

$$value = \frac{(shelfLife - orderAge - decayRate * orderAge * shelfDecayModifier)}{shelfLife}$$

`shelfDecayModifier` is **1** for single-temperature shelves and **2** for the overflow shelf.

Grading Rubric

You are expected to build a system to handle the above scenarios that fulfills each element of this grading rubric to the best of your ability.

- ❑ Meets all specified requirements from *Challenge Prompt* above
- ❑ Is valid code runnable in a modern Android or iOS dev environment
- ❑ Has appropriate usage of design patterns, concurrency, and data structures
- ❑ Has extendable architecture
- ❑ Has a visual interface that allows **interviewers to *clearly understand and follow your system's operations* as it runs in real-time**. Whenever events occur in your system (order received, picked up, discarded, etc), your system should display a message containing **both** a **description of the triggering event** and a **full listing of shelves' contents**.
- ❑ Has comprehensive unit testing *(covers all major components and flows)*
- ❑ Has production-quality code cleanliness
- ❑ Has production-quality docs on all public functions *(as if someone had to work with your code)*
- ❑ Has a README file that contains
  - ❑ Instructions on how to run and test your code in a local environment
  - ❑ A description of how and why you chose to handle moving orders to and from the overflow shelf
  - ❑ Any other design choices you would like the interviewers to know
- ❑ Includes a screenshot or screen capture of your app running

There is no specific guidance on how long you should spend. Similar to working in industry, what matters is the completeness and quality of your submission regardless of how quickly you complete. We've seen quality submissions done in an afternoon, while other candidates take longer based on their pace. Please focus on meeting the grading rubric above (no need for more than that).

Code Submission

At the end of Deep Dive with interviewers, please zip/tarball your code with build instructions and submit it to us via a GreenHouse link. Please do *not* submit binaries.