Engineering Challenge

Challenge Prompt

Create a real-time system that emulates the fulfillment of delivery orders for a kitchen. The kitchen should receive 2 delivery orders per second (see *Appendix* on *Orders*). The kitchen should instantly cook the order upon receiving it, and then place the order on the best-available shelf (see *Appendix* on *Shelves*) 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, framework, and IDE to demonstrate your best work; *however*, we ***strongly discourage*** the use of microservices, kafka, REST APIs, RPCs, DBs, etc due to time constraints.

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 runnable code *(via CLI or within your IDE)*
- ❑ Has appropriate usage of design patterns, concurrency, and data structures
- ❑ Has extendable architecture *(to allow for additions/changes during the Deep Dive with interviewers)*
- ❑ Has console or log output 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 output 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)*

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.

Appendix

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.

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

You must fully complete every element of the Coding Challenge Grading Rubric ***before*** working on this section.

Extra Credit

This **optional** section contains another feature for your system to support. Make sure your system's output is updated to account for additional elements and data. All *Grading Rubric* elements (tests, docs, etc) are applicable to this section as well.

Shelf Life

Instead of orders persisting indefinitely on a shelf, now consider that 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
```

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

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