Engineering Challenge

Build a production-level system that emulates the fulfillment of food orders in kitchen. Your system should allow orders to be placed and dispatched in real time. In this system, orders are prepared by a kitchen and placed on a shelf for drivers to pick up for delivery.

Implementation Details

Create classes as you see fit, each having appropriate encapsulation, attributes, and well defined interfaces.

In a production system, this would be connected to a server and stream orders continuously. For this challenge, orders can be asynchronously read from a json file.

Orders are defined as:

```
[
  {
    "name" : "Cheese Pizza",
    "temp" : "hot",
    "shelfLife" : 300,
    "decayRate" : 0.45
  },
  …
  …
]
```

Where shelf life is read in second units and the temperatures are hot, cold, and frozen.

The kitchen should receive orders at rate following a Poisson Distribution with an average of 3.25 deliveries per second (lambda), make the order (instant), then place the order on its correct shelf.

There are four types of shelves:

- A hot shelf that can store 15 hot orders
- A cold shelf that can store 15 cold orders
- A frozen shelf that can store 15 frozen orders
- An overflow shelf that can store 20 of any order

When the hot, cold, or frozen shelves are full, orders can be placed on the overflow shelf. If all the shelves including the overflow shelf are full then an order should be removed and considered waste. If shelves free up, you may move an order back from the overflow shelf.

Orders that are on a shelf decay (lose value) over time. The value can be calculated as:

```
value = ([shelf life] - [order age]) - ([decay rate] * [order age])
```

Orders that have reached a value of zero are considered waste and should be removed from the shelves. Decay rates double for orders on the overflow shelf. A normalized value for the order is the current value divided by the shelf life of the order.

Drivers should be dispatched to pick up each order as it is placed. Due to distance and traffic, your system should randomly assign a drive time of 2 to 10 seconds for the driver to arrive to pick up the order.

Display the contents of each shelf including the normalized value of each order. The display should be updated every time an order is added or removed.

Deliverables

Please take your time to delivering a quality solution that shows your ability. Include:

- A README file that contains:
    - Instruction 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

- Production-ready code that:
    - Follows community standard syntax and style
    - Has no debug logging, TODOs, or FIXMEs
    - Has test coverage to ensure quality and safety

Please make sure that you've showed us your best code before mimicking production-like infrastructure.

Extra Credit

- Abstract the order decay formula so that it can be dynamic per order.

What we look for

This challenge is meant to help us see your best code, and to showcase your judgment. When we evaluate the challenge, we look at how focused you were on meeting the requirements, at the simplicity and correctness of your architecture, at your use of appropriate design patterns, your choices of threading and data structures, and your use of best practices for code, testing, and documentation.