Engineering Challenge

Build a set of classes that represents the fulfillment of food orders in a kitchen-delivery system where a kitchen produces orders that are 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. All orders should be read from a json file containing an array 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.

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.

Driver should be dispatched to pick up the food order from the kitchen in the same order in which it was received. Due to distance and traffic, each order should be picked up by the driver on a random time scale between 2 and 10 seconds after the order is placed.

Display the contents of each shelf including the normalized value (between 0 and 1) 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
    - A description of how and why you chose to handle overflow
    - A description of how and why you chose to handle underflow

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

Extra Credit

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