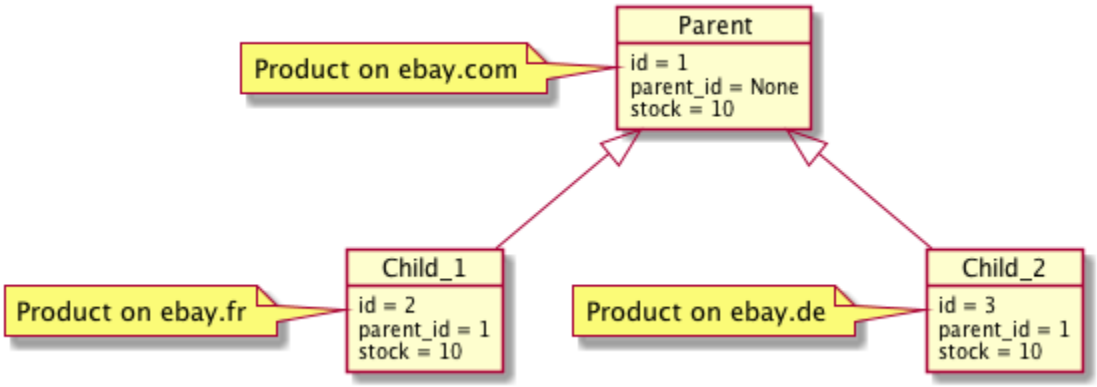
Senior Python Developer - homework

"Stock synchronization application"

Goal of application is to keep products stock in sync (i.e. ebay or amazon products). It means that stock level on all products from one tree should be exactly the same all the time.

There can be multiple parent products, and multiple child products. Each child product has exactly one parent product. Each parent product can have 0 or many child products. One product tree is parent product with it's child products. Imagine that you are selling MS DOS Diskettes in ebay.com, and you have 10 of them in stock. Then same product is also listed in ebay.fr and ebay.de. Those products together creates product tree (example diagram below). Role of application is to keep those item stock in sync (i.e. keep exactly same stock level across all products in tree, when you sell one diskette on ebay.com, stock on ebay.fr and ebay.de should be decreased by 1 and equal 9 on all products).



Product can be ended, it means that product becomes inactive and is not available for buyers. Ending product doesn't change its stock.

Application achieves its goal by consuming and generating events. Each input event can generate 0 or more output events. Each input event has timestamp field which is timestamp of event creation. To make life easier, timestamp is unique across all events. In events containing stock field, value of that field is always current available stock value (i.e. 'stock': 10 means that there is 10 pieces of product available). You should expect receiving incorrect events (wrong type, data missing, wrong data types etc). Output events are events that would trigger changes on products (i.e. commands) in real world (like update product stock or end product). In our application those events are just stored as an output. Imagine that those events would have immediate impact on processed products.

Application should process events one by one in same order as input events are delivered.

Input is a file containing input events represented as JSON, in each line there is one event. Events order in file is order of which application receives events. Order of events creation can be different than order of delivering events. Each input event has timestamp field which is timestamp of event creation.

Output should be a file containing output events represented as JSON, one event per line. Events should be written to file in order of publishing those events by application. At last line of output file there should be JSON with StockSummary event showing snapshot of all product stocks.

Synchronization rules

- when stock goes to 0 in any product from tree, all other products should be ended (but not product with 0 stock),
- when parent product is ended, all child products should be ended
- when child product is ended, nothing happens
- when stock is updated in any product in product tree it should be then updated in other products

Input event types

type field is string, other fields are int

Event name	When triggered	Payload
ProductCreated	when new product is created (product becomes to be active)	{'type': 'ProductCreated', 'id': 123, 'stock': 10, 'timestamp': 123, 'parent_id': None/456} When parent_id is None, then it's parent product.

ProductUpdated	when product stock is updated (increased or decreased)	{'type': 'ProductUpdated', 'id': 123, 'timestamp': 123, 'stock': 10}
ProductEnded	when product is ended (product is inactive)	{'type': 'ProductEnded', 'id': 123, 'timestamp': 123}

Output event types

type field is string, other fields are int except 'stocks' which value is dict of int keys, and int values

Event name	When triggered	Payload
UpdateProduct	when product needs to be updated	{'type': 'UpdateProduct', 'id': 123, 'stock': 10}
EndProduct	when product needs to be ended	{'type': 'EndProduct', 'id': 123}
StockSummary	last generated event, snapshot of stocks state	{'type': 'StockSummary', 'stocks': {'123': 10, '456': 20, '789': 30 ...}} stocks dictionary keys are all products ids, values are current stock level.

Example scenario

Given: I have one parent product (P) with stock 10, and two child products (C1, C2) with stock 10
When: Stock in product P is changed to 9 Then: C1 stock should be updated to 9 Then: C2 stock should be updated to 9
When: Stock in product C1 is changed to 0 Then: Product P should be ended Then: Product C2 should be ended
When: Product C2 is ended Then: Nothing should happen
When: Product P is ended Then: Product C1 should be ended Then: Product C2 should be ended

Input and output examples

Example input file:

```
{'type': 'ProductCreated', 'id': 1, 'stock': 10, 'timestamp': 123,
'parent_id': None}
{'type': 'ProductCreated', 'id': 2, 'stock': 10, 'timestamp': 124,
'parent_id': 1}
{'type': 'ProductCreated', 'id': 3, 'stock': 10, 'timestamp': 125,
'parent_id': 1}
{'type': 'ProductUpdated', 'id': 1, 'timestamp': 126, 'stock': 9}
{'type': 'ProductUpdated', 'id': 1, 'timestamp': 127, 'stock': 8}
{'type': 'ProductUpdated', 'id': 1, 'timestamp': 129, 'stock': 20}
{'type': 'IncorrectEventType', 'id': 1, 'timestamp': 130, 'stock': 20}
// incorrect event
{'type': 'ProductUpdated', 'id': 1, 'timestamp': 131} // incorrect event
{'type': 'ProductUpdated', 'id': 4, 'timestamp': 132, 'stock': 20} //
event for not existing item
{'type': 'ProductUpdated', 'id': 1, 'timestamp': 128, 'stock': 7}
{'type': 'ProductUpdated', 'id': 3, 'timestamp': 134, 'stock': 19}
{'type': 'ProductEnded', 'id': 2, 'timestamp': 135}
{'type': 'ProductUpdated', 'id': 3, 'timestamp': 136, 'stock': 0}
```

Example of output file:

```
{'type': 'UpdateProduct', 'id': 2, 'stock': 9}
{'type': 'UpdateProduct', 'id': 3, 'stock': 9}
{'type': 'UpdateProduct', 'id': 2, 'stock': 8}
{'type': 'UpdateProduct', 'id': 3, 'stock': 8}
{'type': 'UpdateProduct', 'id': 2, 'stock': 20}
{'type': 'UpdateProduct', 'id': 3, 'stock': 20}
{'type': 'UpdateProduct', 'id': 1, 'stock': 19}
{'type': 'UpdateProduct', 'id': 2, 'stock': 19}
{'type': 'EndProduct', 'id': 1}
{'type': 'EndProduct', 'id': 2}
{'type': 'StockSummary', 'stocks': {'1': 19, '2': 19, '3': 0}}
```

Constraints

- input file won't be bigger than 10Mb
- use Python 3.6.0+
- don't use any external dependencies, only python standard library is allowed

Bonus task

- How would you design architecture for system supporting synchronization of hundreds of millions of items in near real time? How would you deploy your solution, what components (like databases, or cache) would be required in your architecture?
- Describe how would you implement asynchronous processing in multiple threads/processes of input events (in contrast to processing one by one).