Best Item Problem

Data Structures

Not surprisingly our core data structures revolve around LEGO® bricks. The real data model is surprisingly complex, but to keep it simple for this example we can imagine the data model consisting of:

- **Brick**: the main unit of building. Contains an integer shape ID and multiple integer color codes. The order of the color codes is important, a brick with color codes {21, 1} is different from one with {1,21}

- **Item**: the business entity we use when manufacturing LEGO® models. Simply put an Item represents what bricks you get for a single price. Contains an integer 'item identifier' and a list of bricks. The order of the bricks is not important.

An item containing ten bricks

An item containing one brick

- **Master Data:** data associated to an Item. Contains the 'item identifier', a price, and a status. Status has four possible values: Novelty, Normal, Outgoing, and Discontinued.
- **Model:** a collection of Bricks with associated 3D position and orientation information

Background

LEGO® has been producing bricks for a long time; this means there are multiple 'Items' representing what is visually the same brick. When an 'Item' is 'Discontinued' it is never made 'Normal' again, instead when we want to start producing the brick again a new 'Item' is created. We don't want to just remove 'Discontinued' 'Items' from our dataset because we want to support viewing older models and still have accurate information.

Our business rules dictate that new models should be made with items that have first the best status and second the lowest price. Status Normal is better than Novelty, which is better than Outgoing. Discontinued items should not be used for new models.

Problem

A Model Designer needs to deliver a list of 'Item Identifiers', alongside the digital LEGO® Model they create, to their Manufacturing Specialist colleague. However, having multiple 'Items' representing what is visually the same brick, combined with complicated business rules, makes it very hard for them to pick the best item to put in their digital model, and this is the problem we need to help them with! ***How would you solve it?***

Expectations

1. Runnable functions that do the 'best item' calculation with a focus on:
   - testability
   - data structures
   - algorithms
   - performance

   Keep it simple, we do not expect a whole GUI application – simply SOLID testable functions

   There could be some difficult corner cases in your approach to the problem – if you run out of time from an implementation point of view, please describe the scenarios in your unit tests so it is clear you are aware of them from a critical thinking perspective

2. C4 container diagram showing how your functions could be used in an end-to-end solution to our model designer's problem

3. You spend less than 6 hours

4. You return your assignment to us at the latest two working days before your third interview e.g., if your interview is sometime Wednesday, we should have your assignment in our Inbox by 8:00 am Monday morning

5. You will spend 30 minutes presenting your case at the final interview, think of it as a design discussion / code review with your potential new team

Input

As input to solving the problem, assume you are given two datasets:

- **`MasterDataList`**: A list of all Master Data.
- **`ItemList`**: A list of all Items.

For C++ based solutions we have included a small project template to get you started based on the Catch2 v2.13.9 header only test library.