Théotime Soulier-Verschaeve

TP#5: Good practices, Refactoring, Testing, API Call

Part 1 :

1.

The aim of this code is to manage the stock of the Guilded Rose by updating some characteristics (Name, SellIn and Quality) of the items that compose the stock. More precisely, the Guilded Rose is represented by a class that has a list of items to sell as attribute (also represented by a class), and this class is composed of method that serve to update the items characteristics.

2.

All the items listed in the Guilded Rose are created in the program and are the following:

- +5 Dexterity Vest
- Aged Brie
- Elixir of the Mongoose
- Sulfuras, Hand of Ragnaros
- Backstage passes to a TAFKAL80ETC concert
- Conjured Mana Cake

3.

At the end of each day, some items characteristics are updated depending on their name, their sellIn and quality value.

#Objects in General

It is possible to split the items in two categories: the classical items and special items.

The classical items see their quality and sellIn value decreasing every day. The quality must be between 0 and 50 and once the sellIn is under 0 (expiration date crossed), this quality decreases twice faster.

#Special objects

They are also several items with unique properties. One of them has a constant quality and no expiration date (Sulfuras), others see their quality increase with time (Aged Brie or Backstage passes), one see its quality increase with time but becomes 0 when the expiration date is crossed (backstage passes) and some items are conjured and see their quality decrease twice faster than normal items.

4.

According to the specifications, the quality of the cheese increase with time and the quality maximum is 50. Then when a day is over, the quality of the cheese will increase if the it is not equal to 50 without exceeding 50 or it will remain at 50.

5.

Once a concert ticket reached its expiration date, its quality is equal to 0.

6.

#No regularity in object instantiation:

The instantiations are not done in the same way and that makes the code less readable.

#The program is not enough decomposed in functions:

The number of functions is to small and there is to much code in them. The code is then less readable and less scalable.

#The variable names are not clear:

The variable names are difficult to read.For example the variable "app" is very blur, it is difficult for someone who reads the code for the first time to understand the meaning and utility of this variable. Same for the index "i".

#Function UpdateQuality is difficult to read:

The function should be decomposed in smaller functions with a very specific goal. Here, the list of if statements make the code unreadable. It is hard to know in which case we are and what is happening.

7.

No because, even if the principal concept is explained, there is an important lack of information about some items behaviour, for example, we don't know how the quality of a Conjured Aged Brie is expected to evolve. The expiration date concepts is also not very clear: There is a sellIn variable than could refer to the expiration date but it is a bit blur.

Part 2:

The benefit of adding test is to decompose as much a possible the tests in smaller test. This allows to know more quickly and with more precision where a problem is when a test fail.

Part 4:

- Adding a storage feature :

This feature would allow to organise the Gilded Rose's stock by adding functionalities that would allow to know how many items of each categories are in the storage, tout count them, to remove, add or access the items.

I would represent this feature by a class. This class would replace the Items list attribute of Gilded Rose's class.

The storage class would have a dictionary as attribute containing the item name as key and a list of items of one category as value.

In addition of the properties, the class would have different methods that would allow to add, remove or count the items present in the storage.

This functionality is interesting because it allows to have more information on the number of products depending on their types and the item storage would be more ordered. It is scalable and allows to add or remove objects more easily and the creation of new types of items is not a problem.

- Adding a transactionnal feature :

This feature would allow to add a financial aspect to the Gilded rose. Each item would have a price and this price could change regarding the quality of the sellIn value (there could be discounts for example). I imagine the Gilded Rose having a certain turnover and benefit. Each product would have a buying and a selling price as attribute and the Gilded Rose could have the possibility to create a stock by buying products and then to sell them at a higher price. Some of the product could be lost after the expiration date and affect the turnover.

To do this, I would create a class GildedRoseTransactions and this class would only contain methods such as buying an item, selling an item and calculate_item_price depending on the category, the quality and the sellIn values.

This functionality is quite scalable too. It can be used on every type of items and be improved with more functionalities.

- Adding a statistical feature :

This functionnality would allow to calculate different interesting statistics about the Gilded Rose such as the number of sales per month, the favourite item of the clients, or some statistics about each article (a function could allow to know at which price the concert ticket is the most sold for example).

To implement this functionality, I would create a class GildedRoseStatistics. This class would contains the methods allowing to calculate the wanted statistics.