TP 5: Gilded Rose – Good practices, Refactoring, Testing

Part 1: Questions

Question 1:

This code is about Gilded Rose which is a store selling items. This code acts as a stock inventory. For each day, we can see the items they have in their inventory and the information about the items such as the "sellIn" and the "quality". The "sellIn" information corresponds to the number of days before the item must be sold. The "quality" information corresponds to a number which indicates how much the item is valuable. There are multiple parameters to take in: the item, the "sellIn" parameter, the "quality" parameter, and the day.

Question 2:

Yes, we can clearly identify the name of all the goods stored in the Guilded Rose. We need to look on the "Program.cs" file and the list of Item object that is created.
We can enumerate 8 working items and 1 not-working item. Among the 9 items, we have some items in multiple copies with different "sellIn" and "quality" parameters.
In the store, we have:

- 1 "+5 Dexterity Vest"
- 1 "Aged Brie"
- 1 "Elixir of the Mongoose"
- 2 "Sulfuras, Hand of Ragnaros" with the same "quality" parameter and different "sellIn" parameter
- 3 "Backstage passes to a TAFKAL80ETC concert" with different "sellIn" and "quality" parameters
- 1 "Conjured Mana Cake" which is not working in Part 1

Question 3:

When the day is over, the "quality" and the "sellIn" of the items are updated. The updates appear on the day after.
The method "UpdateQuality()" modifies the "quality" parameter as well as the "sellIn" parameter.
The method "UpdateQuality()" is called after the "Console.Writeline()" of the item in the loop of the days.

Question 4:

When the day is over, the "quality" of the cheese is increased by 1 or 2. The "quality" of the cheese depends on its "sellIn" parameter. When the day is over, there are two cases:

- When the "sellIn" parameter is equal or greater than 0 and strictly less than 50, the "quality" parameter is increased by 1 for each day.
- When the "sellIn" parameter is strictly less than 0, the "quality" parameter is increased by 2 for each day.

Question 5:

When a concert ticket goes over its expiration day, the "quality" parameter drops to 0. In the "UpdateQuality()" method, the "quality" is deducted by itself to be equal to 0.

Question 6:

The first thing that makes this code hard to read is the “UpdatedQuality()”. The fact the method is constructed on “if” and “else” statements testing the different parameters makes this code hard to read. Some “if” and “else” statements are so long we can’t clearly identify their ends. This function is also really hard to read because there are no comments.
Moreover, there is too much testing on the “name” of the item which makes the code difficult to read and to follow. The code is hard to read because we can’t clearly read the updates for a single item. We need to run through all the method to search for the update on the item we are interested in. The update on the “sellIn” parameter is also hard to find due to all the “if” statements and testing on “quality” and “name” parameters.

The second thing that make this code hard to read is the tests. There are only two tests which can’t test all the features of the code. Moreover, there are no comments in the different tests.

The last thing that can be criticize is the creation of the list in the “Main” function. It could be easier to initialize the Item objects with the same visual, instead of the two visuals in the present code. This part is not the most important thing that makes the code hard to read.

Question 7:

Yes, I think it could be possible to rework the entire solution from scratch. I think the rules on the parameters are clear enough.

Part 2: Testing

The benefit of adding tests here is to use them during the refactoring to be sure that the code is still working after some changes. The tests can be used to assess the updates of the code and validate our changes.

Part 4: Up, up, and away

The first functionalities we can implement an entry of stock of an item directly by the user in a user interface. This could be done with the creation of a constructor for the Item object.

We can create 2 methods in the “Item” class to decrease and increase the quality according to its value (higher than 0 and less than 50). With the inheritance, every classes could have these methods. I didn’t do it in the Part 3 because the updates were only for the “GildedRose.cs” file. I think, by doing that, my function “new_updateQuality() could be smaller and easier to read.