

Presentado por: Andrés Castrillón Velásquez
Santiago Tamayo López

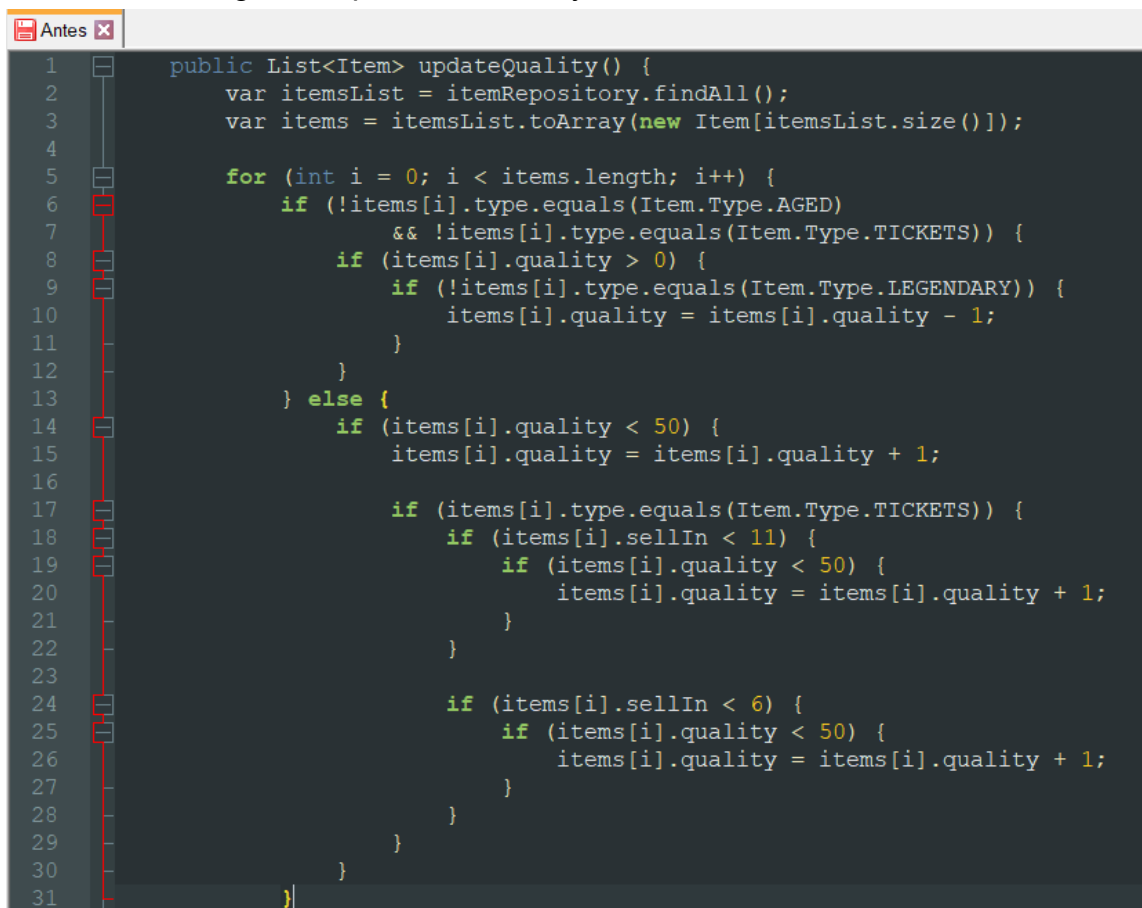
Reformatting Gilded Rose Program

We changed the `updateQuality()` method in the `ItemService` class and added 7 new methods following the Clean Code guidelines and SOLID principles

Conditional Nesting

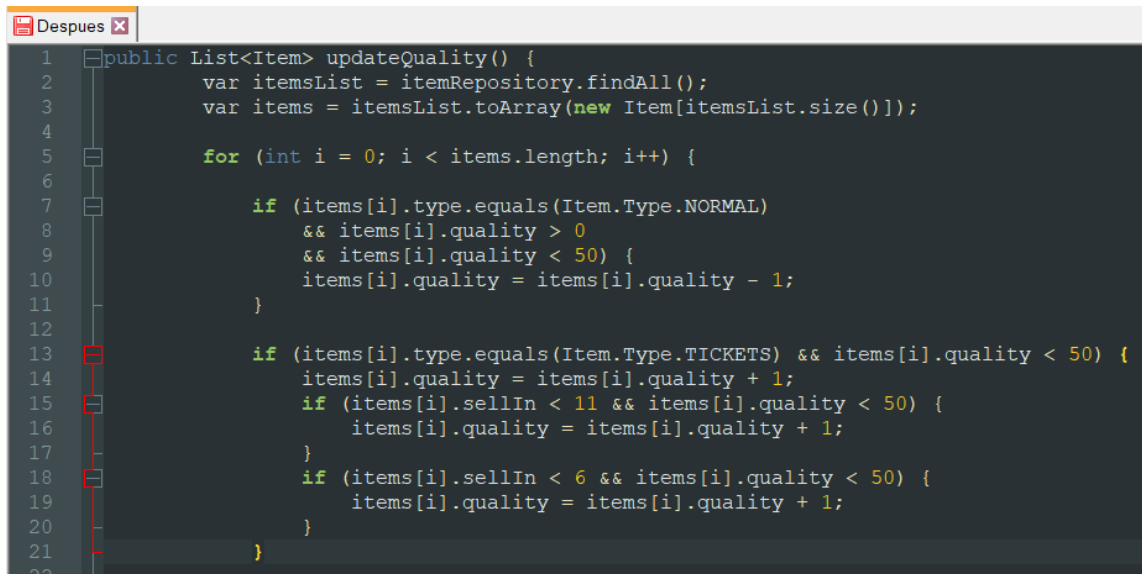
The original method had several conditional nests and also negative conditionals which made it difficult to read.

The first step was separating the conditionals and replacing them with simple ones with the same logic to improve readability



The screenshot shows a code editor window titled 'Antes' (Before) displaying the original `updateQuality()` method. The code is written in Java and features several nested `if` statements and a `for` loop. The logic is as follows:

```
1 public List<Item> updateQuality() {
2     var itemsList = itemRepository.findAll();
3     var items = itemsList.toArray(new Item[itemsList.size()]);
4
5     for (int i = 0; i < items.length; i++) {
6         if (!items[i].type.equals(Item.Type.AGED)
7             && !items[i].type.equals(Item.Type.TICKETS)) {
8             if (items[i].quality > 0) {
9                 if (!items[i].type.equals(Item.Type.LEGENDARY)) {
10                    items[i].quality = items[i].quality - 1;
11                }
12            }
13        } else {
14            if (items[i].quality < 50) {
15                items[i].quality = items[i].quality + 1;
16            }
17
18            if (items[i].type.equals(Item.Type.TICKETS)) {
19                if (items[i].sellIn < 11) {
20                    if (items[i].quality < 50) {
21                        items[i].quality = items[i].quality + 1;
22                    }
23                }
24
25                if (items[i].sellIn < 6) {
26                    if (items[i].quality < 50) {
27                        items[i].quality = items[i].quality + 1;
28                    }
29                }
30            }
31        }
32    }
33 }
```

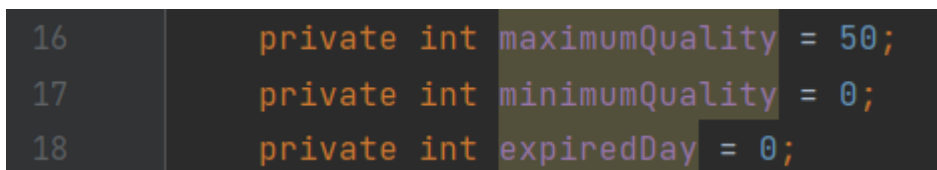


```

1 public List<Item> updateQuality() {
2     var itemsList = itemRepository.findAll();
3     var items = itemsList.toArray(new Item[itemsList.size()]);
4
5     for (int i = 0; i < items.length; i++) {
6
7         if (items[i].type.equals(Item.Type.NORMAL)
8             && items[i].quality > 0
9             && items[i].quality < 50) {
10             items[i].quality = items[i].quality - 1;
11         }
12
13         if (items[i].type.equals(Item.Type.TICKETS) && items[i].quality < 50) {
14             items[i].quality = items[i].quality + 1;
15             if (items[i].sellIn < 11 && items[i].quality < 50) {
16                 items[i].quality = items[i].quality + 1;
17             }
18             if (items[i].sellIn < 6 && items[i].quality < 50) {
19                 items[i].quality = items[i].quality + 1;
20             }
21         }
22     }
23 }

```

After that, we started replacing the constant values for global variables for a better interpretation of the values. We used **Naming conventions** and continued using **camelCase** casing with **Vertical formatting**.

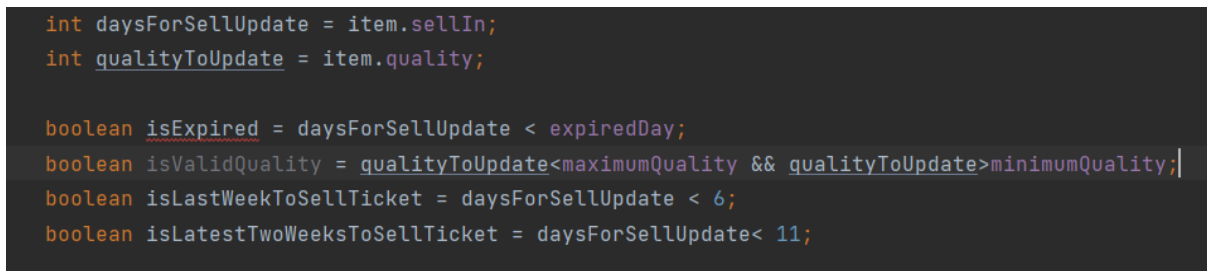


```

16 private int maximumQuality = 50;
17 private int minimumQuality = 0;
18 private int expiredDay = 0;

```

And implemented Boolean operators to improve readability in the conditional checks and inside the methods that use them renamed quality and sellIn attributes.



```

int daysForSellUpdate = item.sellIn;
int qualityToUpdate = item.quality;

boolean isExpired = daysForSellUpdate < expiredDay;
boolean isValidQuality = qualityToUpdate < maximumQuality && qualityToUpdate > minimumQuality;
boolean isLastWeekToSellTicket = daysForSellUpdate < 6;
boolean isLatestTwoWeeksToSellTicket = daysForSellUpdate < 11;

```

Cohesion and Coupling

The 7 Methods are organized by the function it has in the class

```
public int increaseQuality(int qualityToUpdate) throws Exception {...}

public int decreaseQuality(int qualityToUpdate) throws Exception {...}

public int decreaseSellIn(Item item) { return item.sellIn = item.sellIn - 1; }

public int updateTicketQuality(Item item, boolean isExpired) throws Exception {...}

public int updateNormalQuality(Item item, boolean isExpired) throws Exception {...}

public int updateAgedQuality(Item item, boolean isExpired) throws Exception {...}

public Item itemUpdated(Item item, boolean isExpired) throws Exception {...}
```

increaseQuality()

Both TICKETS and AGED items increase their quality over time. so we call this method inside **updateAgedQuality()** and **updateTicketsQuality()**. (S O L D)

- It receives the quality of the item and checks if it is a valid quality then increases it.
- It throws an exception if it tries to increase more than the maximumQuality. (Error Handling)
- Has only one responsibility
- returns the new quality value of the item

```
public int increaseQuality(int qualityToUpdate) throws Exception{
    boolean isValidQuality = qualityToUpdate < maximumQuality && qualityToUpdate > minimumQuality;
    try{
        if(isValidQuality){
            qualityToUpdate++;
        }
        else{
            throw new ArithmeticException("the quality has the maximum value");
        }
    } catch (ArithmeticException e) {
        e.printStackTrace();
    }
    return qualityToUpdate;
}
```

decreaseQuality()

Only Normal items decrease quality, so we call this method inside updateNormalQuality() method. it does the opposite of increaseQuality().

We didn't merge the methods into one like "updateItemQuality()" because it would make it difficult to change the behavior if new types of items are added in the future.
(S O L D)

```
public int decreaseQuality(int qualityToUpdate) throws Exception {
    boolean isValidQuality = qualityToUpdate < maximumQuality && qualityToUpdate > minimumQuality;
    try{
        if(isValidQuality){
            qualityToUpdate --;
        }
        else{
            throw new ArithmeticException("the quality has the minimum value");
        }
    } catch (ArithmeticException e) {
        e.printStackTrace();
    }
    return qualityToUpdate;
}
```

decreaseSellIn()

Self-explanatory method. We made it because it is easier to read. it is called in the itemUpdated() method. **(S O I)**

```
public int decreaseSellIn(Item item){
    return item.sellIn = item.sellIn - 1;
}
```

updateTicketsQuality()

As the name says, it returns the new quality of the Tickets type by checking the days before sellIn, booleans are defined inside it for easier readability. If the item is expired sets quality to the minimum.

It has one purpose, and it doesn't have methods that it doesn't use. (S O I)

```
public int updateTicketsQuality(Item item, boolean isExpired) throws Exception {
    int daysForSellUpdate = item.sellIn;
    int qualityToUpdate = item.quality;

    boolean isLastWeekToSellTicket = daysForSellUpdate < 6;
    boolean isLatestTwoWeeksToSellTicket = daysForSellUpdate < 11;

    qualityToUpdate = increaseQuality(qualityToUpdate);

    if(isLatestTwoWeeksToSellTicket){
        qualityToUpdate = increaseQuality(qualityToUpdate);
    }
    if(isLastWeekToSellTicket){
        qualityToUpdate = increaseQuality(qualityToUpdate);
    }
    if(isExpired){
        qualityToUpdate = minimumQuality;
    }

    return qualityToUpdate;
}
```

updateNormalQuality()

Returns the new quality of the Normal type, if the item has expired, decreases the quality twice. (S O I)

```
public int updateNormalQuality(Item item, boolean isExpired) throws Exception {
    int qualityToUpdate = item.quality;

    qualityToUpdate = decreaseQuality(qualityToUpdate);

    if(isExpired){
        qualityToUpdate = decreaseQuality(qualityToUpdate);
    }

    return qualityToUpdate;
}
```

updateAgedQuality()

It does the opposite of the Normal type, increasing quality per day, and if the item is expired does it twice. We split them to maintain Single Responsibility and Open-Closed Principle in case we want to change Normal or Aged types of items. (S O I)

```
public int updateAgedQuality(Item item, boolean isExpired) throws Exception {
    int qualityToUpdate = item.quality;

    qualityToUpdate = increaseQuality(qualityToUpdate);

    if(isExpired){
        qualityToUpdate = increaseQuality(qualityToUpdate);
    }
    return qualityToUpdate;
}
```

itemUpdated()

We created this method to replace all the conditional checks for the "item.type" in the original code. depending on the type of item calls the appropriate methods explained before, and returns the item with its attributes updated. More cases can be added if any type of item is added. (S O I)

```
public Item itemUpdated(Item item, boolean isExpired) throws Exception {
    switch (item.type){
        case NORMAL -> {
            item.quality = updateNormalQuality(item,isExpired);
            item.sellIn = decreaseSellIn(item);
        }
        case AGED -> {
            item.quality = updateAgedQuality(item,isExpired);
            item.sellIn = decreaseSellIn(item);
        }
        case TICKETS -> {
            item.quality = updateTicketsQuality(item,isExpired);
            item.sellIn = decreaseSellIn(item);
        }
    }
    return item;
}
```

updateQuality() “Original Method”

The original code had 57 lines with 1 for loop and 16 conditional statements.

updateQuality() “New Method”

The new code has 15 lines, it calls the itemUpdated() method and passes the items in the list of items, and the boolean asks if it has expired or not. the itemUpdated() returns the item with the updated attributes and then saves them. **(SOI)**

```
public List<Item> updateQuality() throws Exception {
    var itemsList :List<Item> = itemRepository.findAll();
    var items :Item[] = itemsList.toArray(new Item[itemsList.size()]);

    for (Item item : items) {

        int daysForSellUpdate = item.sellIn;
        boolean isExpired = daysForSellUpdate < expiredDay;

        item = itemUpdated(item,isExpired);

        itemRepository.save(item);
    }
    return Arrays.asList(items);
}
```

TESTS

ItemService

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| • updateQuality() | <div></div> | 100% | <div></div> | 100% | 0 | 3 | 0 | 8 | 0 | 1 |
| • updateTicketsQuality(Item, boolean) | <div></div> | 100% | <div></div> | 100% | 0 | 6 | 0 | 12 | 0 | 1 |
| • itemUpdated(Item, boolean) | <div></div> | 100% | <div></div> | 100% | 0 | 4 | 0 | 10 | 0 | 1 |
| • increaseQuality(int) | <div></div> | 100% | <div></div> | 83% | 1 | 4 | 0 | 8 | 0 | 1 |
| • decreaseQuality(int) | <div></div> | 100% | <div></div> | 83% | 1 | 4 | 0 | 8 | 0 | 1 |
| • updateItem(int, Item) | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| • ItemService(ItemRepository, Item[]) | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 7 | 0 | 1 |
| • updateNormalQuality(Item, boolean) | <div></div> | 100% | <div></div> | 100% | 0 | 2 | 0 | 5 | 0 | 1 |
| • updateAgedQuality(Item, boolean) | <div></div> | 100% | <div></div> | 100% | 0 | 2 | 0 | 5 | 0 | 1 |
| • findById(int) | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| • decreaseSellIn(Item) | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| • createItem(Item) | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| • lambda\$findById\$0() | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| • listItems() | <div></div> | 100% | <div></div> | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 0 of 292 | 100% | 2 of 34 | 94% | 2 | 32 | 0 | 70 | 0 | 14 |

Tests are still with 100% coverage, and the missed branches are because of the exceptions in the increase and decrease quality methods. those should never happen.

Conclusions

After applying SOLID principles and good practices with Clean Code guidelines, the code became more readable, with less redundancy and more consistency. It will also be easier to update and maintain in the future in case we want to add new features.

Tests were not affected by the change, so we can conclude that the functionality of the code and the business rules still apply.

We still think the code can improve more functionality without adding more complexity, and will be updating it as we see fit for the next assignments.