



## LAB 03 : REFACTORING AND TESTING

**Name : Md Muktadir Mazumder**

**Course Title : SWE 4302**

**Student Id : 190042136**

**Department : Software Engineering**

## Report on code Refactoring and Testing

### Overview of Test Class

Before refactoring the Inventory class, several tests have been executed in another package aside main, called package test. In the test package, a class has been created called InventoryTest to check different modules of the class Inventory. Several tests have been carried out using different testing methods to check whether our Inventory class is working out properly according to our logic that came from the problem statement as well as every module in the Inventory class is giving our expected output or not. And this unit testing is done with the help of a unit testing framework, Junit5.

#### Testing methods and their justifications:

##### 1. testingQualityOfUnExpiredNormalItemsWhenQualityDecreasesByOne():

In this testing method, I have checked the quality of unexpired normal items when quality will decrease by one, within the sellIn value.

##### Justification:

From the problem statement, it is known that if an unexpired normal item approaches its expiry date the quality will decrease by one. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Normal Items", sellIn value: 5, and quality: 19 from this test method; according to the problem statement if this sellIn value decreases then the quality of the item will also decrease. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellIn, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfUnExpiredNormalItemsWhenDecreasesByOne() {
    System.out.println("Testing quality Of un-expired normal item when quality decreases by one.");
    Item[] items = new Item[] {
        new Item( name: "Normal Items", sellIn: 5, quality: 19)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Normal Items", application.items[0].name);
    assertEquals( expected: 4, application.items[0].sellIn);
    assertEquals( expected: 18, application.items[0].quality);
}
```

##### 2. testingQualityOfExpiredNormalItemsWhenQualityDecreasesByTwo():

In this test method, I have checked the testing quality of expired normal items when it decreases by value two.

##### Justification:

From the problem statement, it is known that if an expired normal item approach crosses its expiry date the quality will decrease by two. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Normal Items", sellIn value: -4 and quality: 2 from this test method; according to the problem statement if this sellIn value decreases then the quality of the item

will also decrease. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfExpiredNormalItemsWhenDecreasesByTwo() {
    System.out.println("Testing quality of expired normal item when decreases by two.");
    Item[] items = new Item[] {
        new Item( name: "Normal Items", sellin: -4, quality: 2)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Normal Items", application.items[0].name);
    assertEquals( expected: -5, application.items[0].sellIn);
    assertEquals( expected: 0, application.items[0].quality);
}
```

### 3. testingQualityOfUnexpiredItemAgedBrieWhenQualityIncreasesByOne():

After the testing of normal items, I have tested the quality of unexpired Aged Brie item when its quality increases by one.

#### Justification:

From the problem statement, it is known that if an unexpired Aged Brie item approaches its expiry date the quality will increase by one. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Aged Brie", sellIn value: 5, and quality: 3 from this test method; according to the problem statement if this sellIn value decreases then the quality of the item will increase by one. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfUnexpiredItemAgedBrieWhenIncreasesByOne() {
    System.out.println("Testing quality of un-expired item 'Aged Brie' when increases by one.");
    Item[] items = new Item[] {
        new Item( name: "Aged Brie", sellin: 5, quality: 3)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Aged Brie", application.items[0].name);
    assertEquals( expected: 4, application.items[0].sellIn);
    assertEquals( expected: 4, application.items[0].quality);
}
```

#### 4. testingQualityOfExpiredItemAgedBrieWhenQualityIncreasesByTwo():

In this test method, I have tested the quality of expired Aged Brie items when its quality increases by two.

##### Justification:

From the problem statement, it is known that if an expired Aged Brie item crosses its expiry date the quality will increase by two. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Aged Brie", sellIn value: -1 and quality: 4 from this test method; according to the problem statement if this sellIn value decreases then the quality of the item will increase by two. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfExpiredItemAgedBrieWhenIncreasesByTwo() {
    System.out.println("Testing quality of expired item 'Aged Brie' when increases by two.");
    Item[] items = new Item[] {
        new Item( name: "Aged Brie", sellin: -1, quality: 4)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Aged Brie", application.items[0].name);
    assertEquals( expected: -2, application.items[0].sellIn);
    assertEquals( expected: 6, application.items[0].quality);
}
```

#### 5. testingQualityOfUnExpiredItemAgedBrieWhenQualityEqualMaxQualityThreshold():

In this method, I have tested to check an unexpired Aged Brie item when its quality is equal to the maximum quality threshold.

##### Justification:

From the problem statement, it is known that the quality of Aged Brie Item will never cross its quality value, fifty. Therefore its maximum quality always remains fifty. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Aged Brie", sellIn value: 5, and quality: 50 from this test method. So, we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfUnExpiredItemAgedBrieWhenQualityEqualMaxQualityThreshold() {
    System.out.println("Testing quality of un-expired item 'Aged Brie' when quality is equal to the maximum quality threshold.");
    Item[] items = new Item[] {
        new Item( name: "Aged Brie", sellin: 5, quality: 50)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Aged Brie", application.items[0].name);
    assertEquals( expected: 4, application.items[0].sellIn);
    assertEquals( expected: 50, application.items[0].quality);
}
```

## 6. testingQualityOfBackstagePassesItemWhenSellInDaysIsOverTenDaysAndQualityIncreasesByOne():

In this method I have tested the quality of Backstage Passes item when the sellIn days are over ten days and its quality increases by one.

### Justification:

From the problem statement, it is known that if a Backstage Passes item approaches its sellIn value approaches over ten days the quality will increase by one. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Backstage passes to a TAFKAL80ETC concert", sellIn value: 12, and quality: 4 from this test method; according to the problem statement if this sellIn value is over ten days then the quality of the item will increase by one. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfItemBackstagePassesWhenSellInDaysOverTenDaysAndQualityIncreasesByOne() {
    System.out.println("Testing quality of item 'Backstage Passes' when SellIn days are over ten days and quality is increased by one.");
    Item[] items = new Item[] {
        new Item( name: "Backstage passes to a TAFKAL80ETC concert", sellin: 12, quality: 4)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Backstage passes to a TAFKAL80ETC concert", application.items[0].name);
    assertEquals( expected: 11, application.items[0].sellIn);
    assertEquals( expected: 5, application.items[0].quality);
}
```

## 7. testingQualityOfBackstagePassesItemWhenSellInDaysIsBetweenSixAndTenDaysAndQualityIncreases ByTwo():

In this test method, I have tested the quality of Backstage Passes item when sellIn days is between six and ten days and quality increases by two.

### Justification:

From the problem statement, it is known that if a Backstage Passes item's sellIn value is between six and ten days then the quality will increase by two. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Backstage passes to a TAFKAL80ETC concert", sellIn value: 10, and quality: 6 from this test method; according to the problem statement if this sellIn value is between six and ten days then the quality of the item will increase by two. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfBackstagePassesItemWhenSellInDaysIsBetweenSixAndTenDaysAndQualityIncreasesByTwo() {
    System.out.println("Testing quality of item 'Backstage Passes' when SellIn days are between six and ten days inclusive and quality increases by two.");
    Item[] items = new Item[] {
        new Item( name: "Backstage passes to a TAFKAL80ETC concert", sellin: 10, quality: 6)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Backstage passes to a TAFKAL80ETC concert", application.items[0].name);
    assertEquals( expected: 9, application.items[0].sellIn);
    assertEquals( expected: 8, application.items[0].quality);
}
```

## 8. testingQualityOfBackstagePassesItemWhenSellInDaysIsLessThanFiveDaysAndQualityIncreasesByThree():

In this method, I have tested to check the quality of Backstage Passes item when the sellIn days are less than five days and increases by three.

### Justification:

From the problem statement, it is known that if a Backstage Passes item's sellIn value is less than five days then the quality will increase by three. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Backstage passes to a TAFKAL80ETC concert", sellIn value: 4, and quality: 10 from this test method; according to the problem statement if this sellIn value is less than five days then the quality of the item will increase by three. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfItemBackstagePassesWhenSellInDaysLessThanFiveDaysAndQualityIncreasesByThree() {
    System.out.println("Testing quality of un-expired 'Backstage Passes' item when SellIn days are less than five days and quality increases by three.");
    Item[] items = new Item[] {
        new Item( name: "Backstage passes to a TAFKAL80ETC concert", sellin: 4, quality: 10)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Backstage passes to a TAFKAL80ETC concert", application.items[0].name);
    assertEquals( expected: 3, application.items[0].sellIn);
    assertEquals( expected: 13, application.items[0].quality);
}
```

## 9. testingQualityOfExpiredBackstagePassesItemWhenQualityWillBeZero():

In this method, I have tested the quality of expired Backstage Passes item to check it's quality is zero or not after expiration.

### Justification:

From the problem statement, it is known that if a Backstage Passes item is expired then it's quality will be zero. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Backstage passes to a TAFKAL80ETC concert", sellIn value: -1 and quality: 10 from this test method. After that we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfExpiredBackstagePassesAndQualityWillBeZero() {
    System.out.println("Testing quality of expired 'Backstage Passes' item and quality will be zero.");
    Item[] items = new Item[] {
        new Item( name: "Backstage passes to a TAFKAL80ETC concert", sellin: -1, quality: 10)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Backstage passes to a TAFKAL80ETC concert", application.items[0].name);
    assertEquals( expected: 0, application.items[0].quality);
}
```

#### 10. testingQualityOfUnExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems():

Besides, in this test method, I have tested that the unexpired conjured items degrade twice as fast as the normal items.

##### Justification:

From the problem statement, it is known that the unexpired Conjured items usually degrades twice the normal items. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Conjured Items", sellIn value: 6, and quality: 4 from this test method; according to the problem statement the quality of the item will increase by twice the degradation value of a normal item.

```
@Test
public void testingQualityOfUnExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems() {
    System.out.println("Testing quality of un-expired 'Conjured item' when degradation in quality is twice ss fast as normal items.");
    Item[] items = new Item[] {
        new Item( name: "Conjured Items", sellIn: 6, quality: 4)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Conjured Items", application.items[0].name);
    assertEquals( expected: 5, application.items[0].sellIn);
    assertEquals( expected: 2, application.items[0].quality);
}
```

So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellIn, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

#### 11. testingQualityOfExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems():

Besides, in this test method, I have tested that the expired conjured items degrade twice as fast as the normal items.

##### Justification:

From the problem statement, it is known that the expired Conjured items usually degrades twice the normal items. So to test this logic, in the Item array of Inventory class we have passed the name of

the item name: "Conjured Items", sellIn value: -1 and quality: 4 from this test method; according to the problem statement the quality of the item will decrease by twice the degradation value of a normal item. So, to check this we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems() {
    System.out.println("Testing quality of expired 'Conjured item' when degradation in quality is twice as fast as normal items.");
    Item[] items = new Item[] {
        new Item( name: "Conjured Items", sellin: -1, quality: 4)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Conjured Items", application.items[0].name);
    assertEquals( expected: -2, application.items[0].sellIn);
    assertEquals( expected: 0, application.items[0].quality);
}
```

## 12. testingQualityOfLegendaryItemWhenSulfurasToCheckItsQualityIsEightyAndItNeverAlters():

In this method I have tested the quality of legendary item Sulfuras to check whether its quality value never alters and always remains eighty or not.

### Justification:

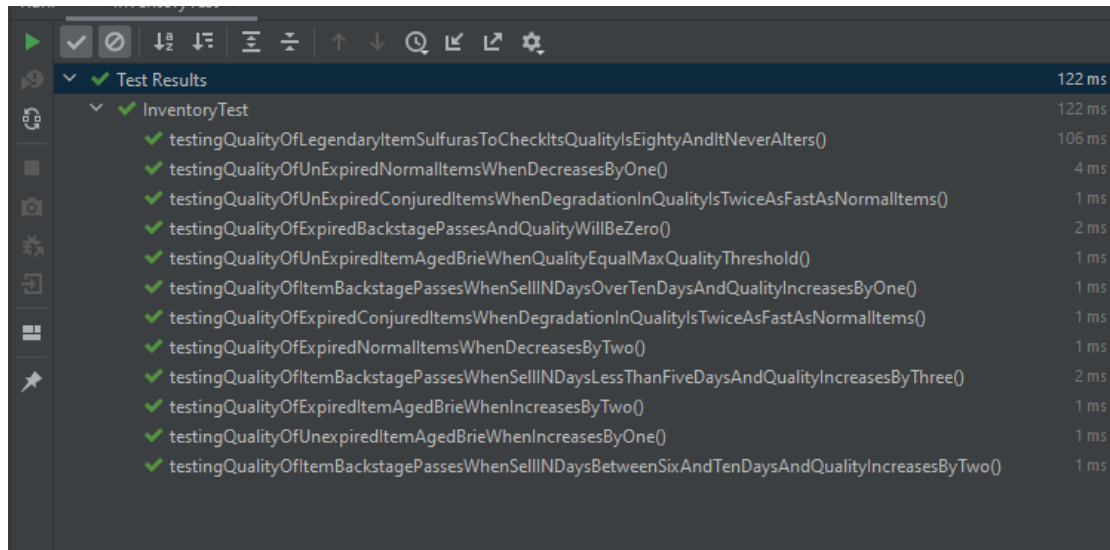
From the problem statement, it is known that the quality of legendary item Sulfuras always remains eighty and it never gets to alter. So to test this logic, in the Item array of Inventory class we have passed the name of the item name: "Sulfuras, Hand of Ragnaros", sellIn value: 5, and quality: 80 from this test method. After that, we have passed our expected value in the assertEquals() method and this method compares our expected result with the passing item name, sellin, and quality values. After the execution of this test, ultimately we have seen this test has been passed, in the test results.

```
@Test
public void testingQualityOfLegendaryItemSulfurasToCheckItsQualityIsEightyAndItNeverAlters() {
    System.out.println("Testing quality of Legendary item 'Sulfuras' to check its quality is eighty and it never alters.");
    Item[] items = new Item[] {
        new Item( name: "Sulfuras, Hand of Ragnaros", sellin: 5, quality: 80)
    };
    Inventory application = new Inventory(items);
    application.updateQuality();
    assertEquals( expected: "Sulfuras, Hand of Ragnaros", application.items[0].name);
    assertEquals( expected: 5, application.items[0].sellIn);
    assertEquals( expected: 80, application.items[0].quality);
}
```



## Results of testing methods:

These are the outcomes of all the test methods when we run our InventoryTest class in the IDE.



Test Results	122 ms
InventoryTest	122 ms
testingQualityOfLegendaryItemSulfurasToCheckItsQualityIsEightyAndItNeverAlters()	106 ms
testingQualityOfUnExpiredNormalItemsWhenDecreasesByOne()	4 ms
testingQualityOfUnExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems()	1 ms
testingQualityOfExpiredBackstagePassesAndQualityWillBeZero()	2 ms
testingQualityOfUnExpiredItemAgedBrieWhenQualityEqualMaxQualityThreshold()	1 ms
testingQualityOfItemBackstagePassesWhenSellInDaysOverTenDaysAndQualityIncreasesByOne()	1 ms
testingQualityOfExpiredConjuredItemsWhenDegradationInQualityIsTwiceAsFastAsNormalItems()	1 ms
testingQualityOfExpiredNormalItemsWhenDecreasesByTwo()	1 ms
testingQualityOfItemBackstagePassesWhenSellInDaysLessThanFiveDaysAndQualityIncreasesByThree()	2 ms
testingQualityOfExpiredItemAgedBrieWhenIncreasesByTwo()	1 ms
testingQualityOfUnexpiredItemAgedBrieWhenIncreasesByOne()	1 ms
testingQualityOfItemBackstagePassesWhenSellInDaysBetweenSixAndTenDaysAndQualityIncreasesByTwo()	1 ms

## Console-output of testing methods:

```
✓ Tests passed: 12 of 12 tests - 122 ms
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ...
<---Running before executing all the tests--->

Testing quality of Legendary item 'Sulfuras' to check its quality is eighty and it never alters.
Testing quality Of un-expired normal item when quality decreases by one.
Testing quality of un-expired 'Conjured item' when degradation in quality is twice ss fast as normal items.
Testing quality of expired 'Backstage Passes' item and quality will be zero.
Testing quality of un-expired item 'Aged Brie' when quality is equal to the maximum quality threshold.
Testing quality of item 'Backstage Passes' when SellIn days are over ten days and quality is increased by one.
Testing quality of expired 'Conjured item' when degradation in quality is twice as fast as normal items.
Testing quality of expired normal item when decreases by two.
Testing quality of un-expired 'Backstage Passes' item when SellIn days are less than five days and quality increases by three.
Testing quality of expired item 'Aged Brie' when increases by two.
Testing quality of un-expired item 'Aged Brie' when increases by one.
Testing quality of item 'Backstage Passes' when SellIn days are between six and ten days inclusive and quality increases by two.

<---Running after executing all the tests--->

Process finished with exit code 0
```

### Use of @BeforeAll and @AfterAll annotations:

Besides all the testing methods in the InventoryTest class, I have used @BeforeAll annotation on the method shouldRunBeforeExecutingAllTheTest(). The annotation usually indicates that this method will be executed before each test in the InventoryTest class.

```
@BeforeAll
public static void shouldRunBeforeExecutingAllTheTest() {
    System.out.println("<---Running before executing all the tests--->");
    System.out.println();
}
```

On the other hand, @AfterAll annotation is used on the method shouldRunAfterExecutingAllTheTest(); to run after each test case. @AfterAll annotation is executed after all the tests in the InventoryTest class.

```
@AfterAll
public static void shouldRunAfterExecutingAllTheTest() {
    System.out.println();
    System.out.println("<---Running after executing all the tests--->");
}
```

## Overview of Refactored Class

Our Inventory class has been refactored after doing the unit testing. We have refactored our Inventory class to make code clean and increase readability. By refactoring, we tried to improve the design of the existing code without changing its observable behavior.

### Iteration 01:

Declared two variables maximumQualityValue and qualityDegradationOfNormalItem to represent the maximum quality threshold and quality degradation for un-expired normal items. It would make it easier to change these values just in one place rather than changing in all places.

### Iteration 02:

The for loop has been replaced with the for-each loop to iterate through elements of the array in Inventory class.

### Iteration 03:

It would be more flexible to check the Sulfuras item condition at the beginning rather than checking at different places. For potential changes regarding this item would be easier in this way.

#### **Iteration 04:**

Rather than using several if-else statements to check the range of the quality value, I have used the max and min functions to check the range. This would make the code concise and readable.

#### **Iteration 05:**

The last segment of the given code was used to check the Sell IN days and used the same if-else condition as the beginning of the code. It is repetition and should be avoided. I have discarded the last segment of the code and merged this part with the beginning part of the code. This makes the code concise and can be changed very easily for future updates.

#### **Iteration 06:**

I think updating the quality value of "Backstage Passes" in several different locations made the code not that readable and a bit complicated. Therefore, I changed the code so that the update of every item should be in a single if or else if segment.

-----THE END-----