Name: Aravind Anand
Student ID: 030821269
Title: Code Refactoring

Question 1: Hi and welcome to team NeighborhoodFresh. We are a small inn with a prime location in a prominent city run by a friendly innkeeper named Allison. We also buy and sell only the finest goods. Unfortunately, our goods are constantly degrading in quality as they approach their sell by date. We have a system in place that updates our inventory for us. The software system was developed by Robert, who has moved on to new adventures. Your task is to add a new feature to our system so that we can begin selling a new category of items. First an introduction to our system:
All items have a SellIn value which denotes the number of days we have to sell the item.
All items have a Quality value which denotes how valuable the item is.
At the end of each day our system lowers both values for every item.

Pretty simple, right? Well, this is where it gets interesting:
Once the sell by date has passed, Quality degrades twice as fast.
The Quality of an item is never negative.
"Aged Brie" actually increases in Quality the older it gets.
The Quality of an item is never more than 50.
"Sulfuras", being a legendary item, never has to be sold or decreases in Quality.
"Backstage passes", like aged brie, increases in Quality as it's SellIn value approaches; Quality increases by 2 when there are 10 days or less and by 3 when there are 5 days or less but Quality drops to 0 after the concert.
We have recently signed a supplier of conjured items. This requires an update to our system:
"Conjured" items degrade in Quality twice as fast as normal items.

Feel free to make any changes to the UpdateQuality method and add any new code as long as everything still works correctly. However, do not alter the Item class or Items property as those belong to Catherine in the corner who doesn't believe in shared code ownership (you can make the UpdateQuality method and Items property static if you like, we'll cover for you). Just for clarification, an item can never have its Quality increase above 50, however "Sulfuras" is a legendary item and as such its Quality is 80 and it never alters.

Refactoring Techniques Used:

The following refactoring is performed for the software system developed by Robert. There were some bad code smells identified and modified with the recommended solutions for the bad smell. Note that there was no alteration made to the Item class.

i. In the source file, the GlidedRose.java file has the Neigborhoodfresh class and if any the other class is created as part of refactoring within the GlidedRose.java would have created multiple class within a single file and it is not a good practice. It suggested creating multiple class files separately and maintaining them under a single package. So, the Neigborhoodfresh class and other classes created as part of refactoring are under the package: com.neigborhoodfresh.

ii. The updateQuality() method of the Neigborhoodfresh class had all the conditions check and modifications for the different types of items implemented using conditional statements (if-else) which could be a bad code smell and was hard to read and understand the logic.

Replace Conditional with Polymorphism:

- This was refactored by implementing Polymorphism using an abstract class - QualityChangingItem that has the abstract method called updateQuality() which is overridden by the classes extending the abstract class based on the alterations performed for each item of different types namely – Aged Brie, Sulfuras, Backstage Passes and Conjured.

iii. Secondly, the common operations such as increasing the quality, decreasing the quality, decreasing the sellin were redundant.

Extract Method:

- It is refactored as methods namely incrementQuality(), decrementQuality(), and decrementSellIn() in the abstract method, and it is called in the extending classes, with some basic conditions check for the respective type of item.

iv. As it not suggested to alter the Item class, the objects to access the respective type of item Class is performed using a wrapping class ItemTypeAccess and object instantiation is performed based on the item type. So, now the updateQuality() method in the NeighborhoodFresh class will instantiate an object for each item based on the condition check for the type of item in the ItemFactory class.

v. Replace Magic Numbers with Variable Constant:

- The maximum quality value (50) and the minimum quality value (0) is set the variables QUALITY_MIN and QUALITY_MAX in the QualityChangingItem class which helps in understanding the overall operation of the Class and this also helps in future modification of the maximum and minimum value of the quality at ease.

vi. Replace Nested Conditional with Guard Clauses:

- While performing refactoring, initially the object instantiation based on the type of Object under ItemTypeAccess class was written with if-else conditional statement, later it was refactored using guard clauses.

Test Cases:

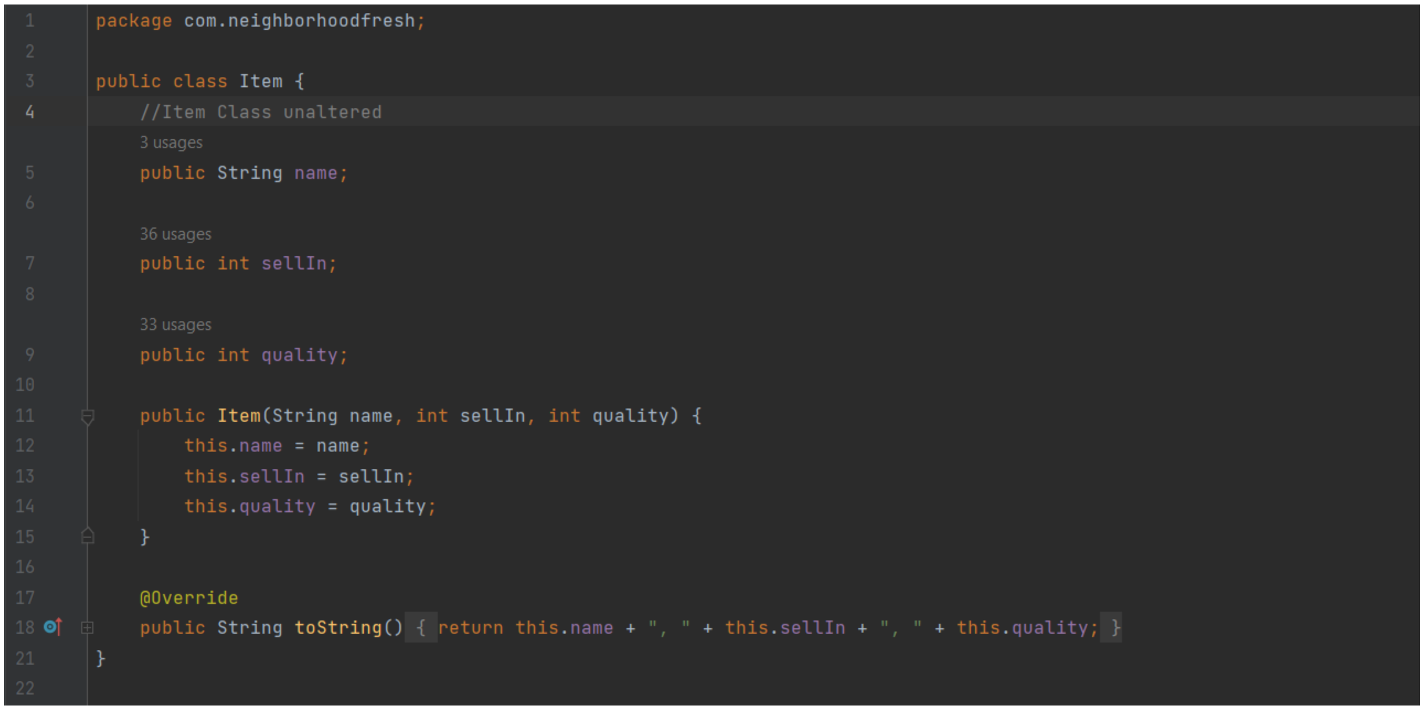
- The refactored code is tested using Java's Junit Testing
- An item of each type categorized, and two new items (treated as default) were considered for the test case and the updateQuality() operation was performed for 31 days for each item.
- The test case will check the asserted values and actual values of quality and sellin for all the items for every 10 days and the test case as **PASSED** successfully.
- Below key behaviors were observed for the given item, sellin value, and quality value in the test case:
    - Aged Brie quality value was increasing by 1 till the sellin value reached 0 and then was increasing by 2. Later, it stopped increasing when the quality value reached 50, this is right as the maximum quality is 50.
    - Sulfuras, Hand of Ragnaros value was unaltered as it is a legendary item.
    - Backstage passes to a TAFKAL80ETC concert, increased the quality value by 1 till the sellin value was 11, when the sellin value reached 10, the quality value increased by 2 for every sellin value decrement and when the sellin value reached 5, the quality value increased by 3 for every sellin value decrement and when the sellin value reached 0, the quality value also reached 0.
    - Conjured is a newly added category and it has decreased as twice as normal, that is for every sellin decrement, the quality value decreased by 2 and when the sellin reached 0 and decreased further, the quality value decreased by 4.
    - Healing Chalice and Winged Boots were newly introduced items to the test case, and they behave as default items. The quality value decrement by 1 until the sellin value reaches 0 and decrease by 2 when the sillin value is below 0 and quality value decreases until it reaches 0.

Also, the same input is fed and checked with the program, before and after refactoring, except Conjured, as this category was added as part of the refactoring. The outputs of the test cases were provided with the submission (named Output_Before_Refactoring_No_Conjured and Output_After_Refactoring_With_Conjured) and they match the same except Conjured. The refactored source code along with the test cases is attached with the submission and code snippets are followed.

Code Snippets:

Item Class (Unaltered):

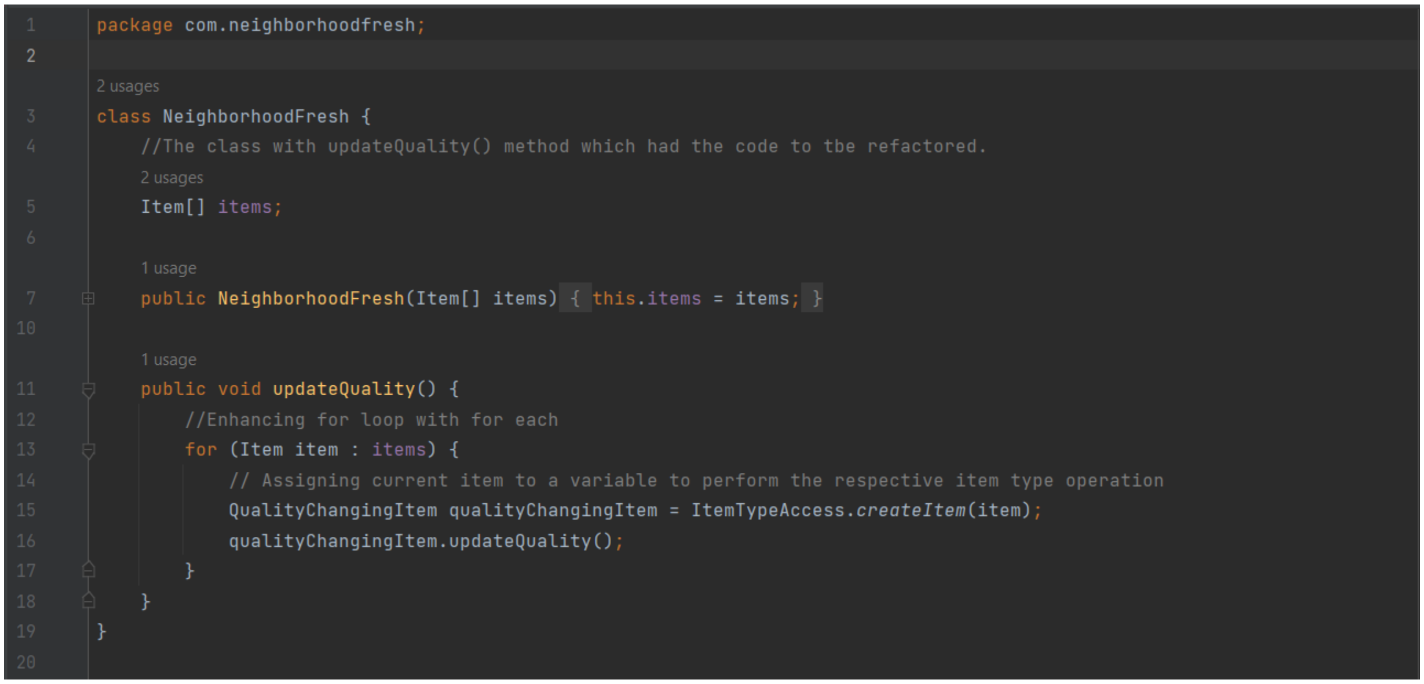
- No changes made to this class.



```java
package com.neighborhoodfresh;

public class Item {
    //Item Class unaltered
    3 usages
    public String name;

    36 usages
    public int sellIn;

    33 usages
    public int quality;

    public Item(String name, int sellIn, int quality) {
        this.name = name;
        this.sellIn = sellIn;
        this.quality = quality;
    }

    @Override
    public String toString() { return this.name + ", " + this.sellIn + ", " + this.quality; }
}
```

NeighborhoodFresh Class:

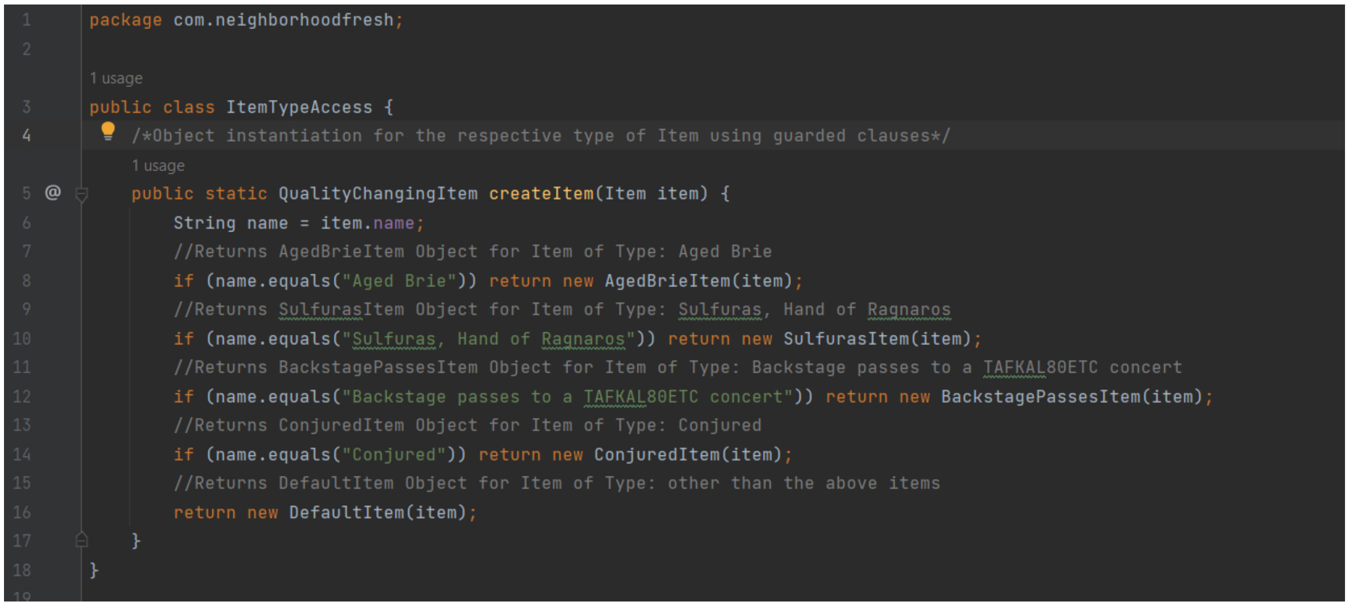
- The conditional statements were refactored to classes for each type of item using polymorphism and each item operation is performed by assigning variable for an item to access their respective class.



```java
package com.neighborhoodfresh;

2 usages
class NeighborhoodFresh {
    //The class with updateQuality() method which had the code to tbe refactored.
    2 usages
    Item[] items;

    1 usage
    public NeighborhoodFresh(Item[] items) { this.items = items; }

    1 usage
    public void updateQuality() {
        //Enhancing for loop with for each
        for (Item item : items) {
            // Assigning current item to a variable to perform the respective item type operation
            QualityChangingItem qualityChangingItem = ItemTypeAccess.createItem(item);
            qualityChangingItem.updateQuality();
        }
    }
}
```

ItemTypeAccess Class:

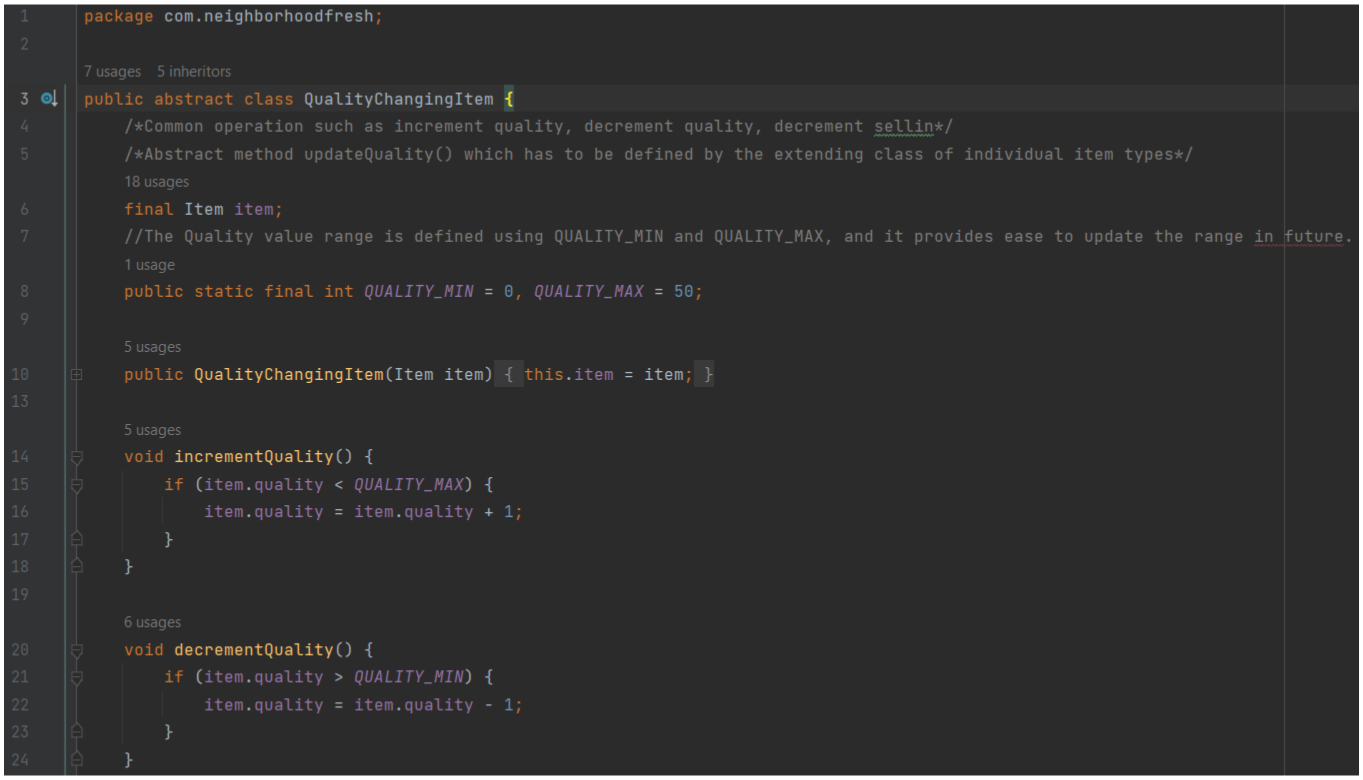
- The class where the object instantiation occurs based on the type of item.



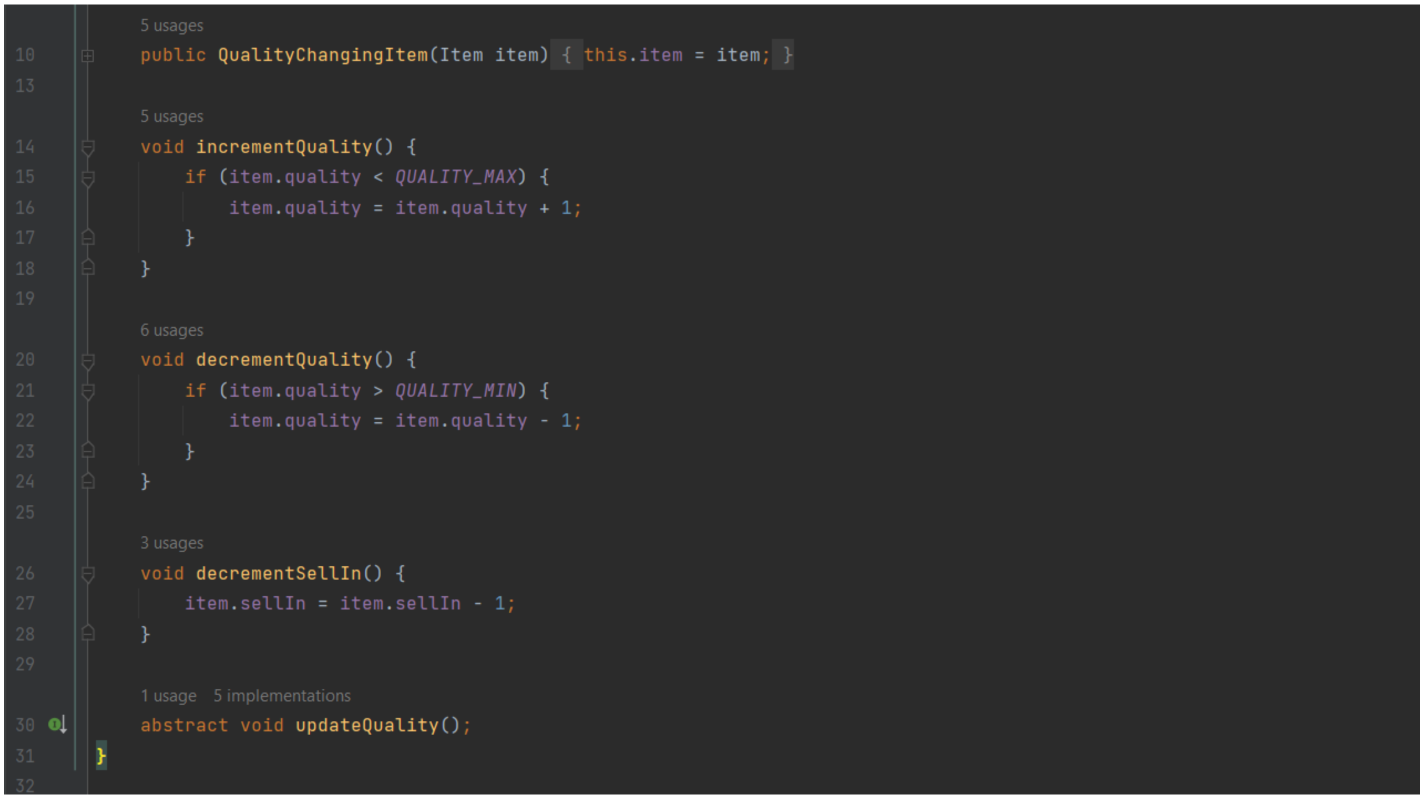
```java
package com.neighborhoodfresh;

1 usage
public class ItemTypeAccess {
    /*Object instantiation for the respective type of Item using guarded clauses*/
    1 usage
    public static QualityChangingItem createItem(Item item) {
        String name = item.name;
        //Returns AgedBrieItem Object for Item of Type: Aged Brie
        if (name.equals("Aged Brie")) return new AgedBrieItem(item);
        //Returns SulfurasItem Object for Item of Type: Sulfuras, Hand of Ragnaros
        if (name.equals("Sulfuras, Hand of Ragnaros")) return new SulfurasItem(item);
        //Returns BackstagePassesItem Object for Item of Type: Backstage passes to a TAFKAL80ETC concert
        if (name.equals("Backstage passes to a TAFKAL80ETC concert")) return new BackstagePassesItem(item);
        //Returns ConjuredItem Object for Item of Type: Conjured
        if (name.equals("Conjured")) return new ConjuredItem(item);
        //Returns DefaultItem Object for Item of Type: other than the above items
        return new DefaultItem(item);
    }
}
```

QualityChangeItem Class:

- Class with all the common operations such as incrementQuality(), decrementQuality(), decrementSellin() and abstract method updateQuality() which will be defined by the extending class of each type.



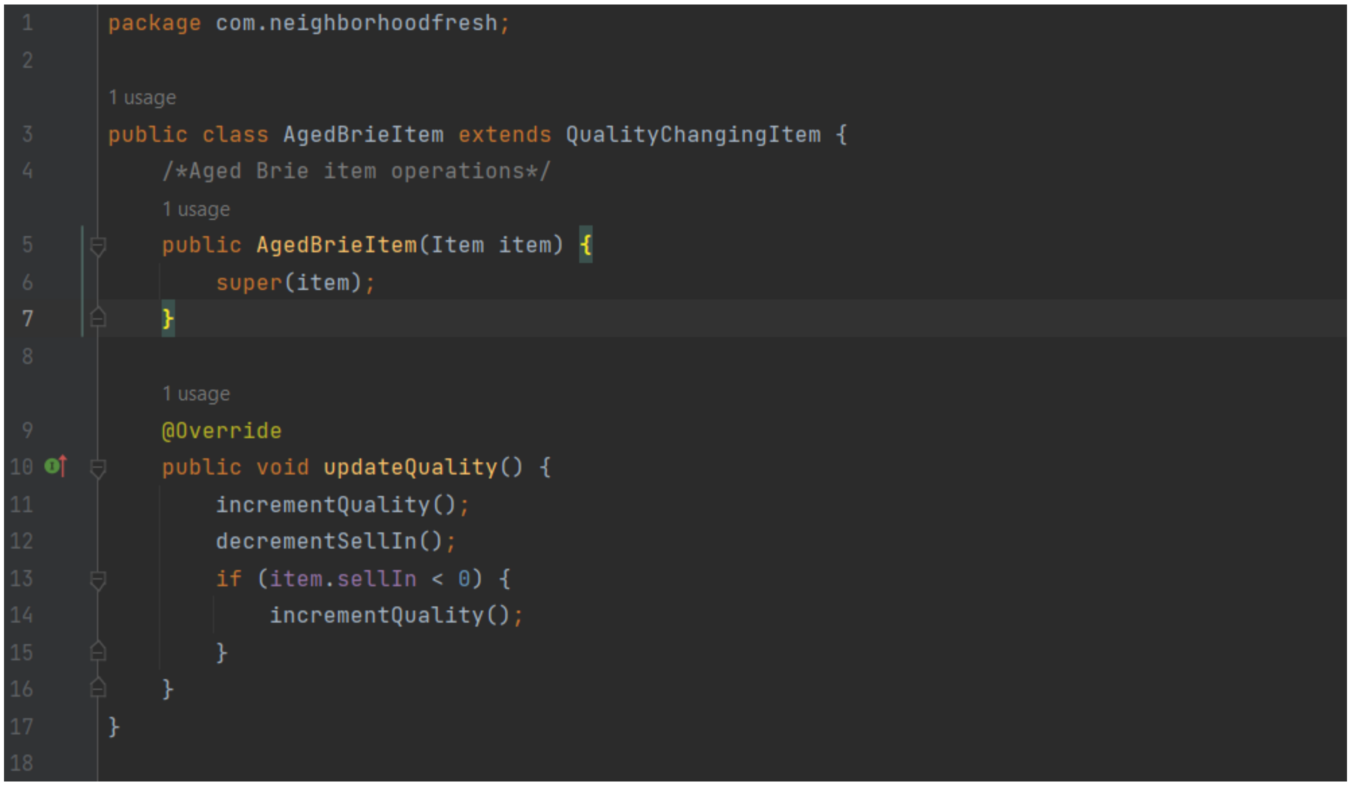
```java
package com.neighborhoodfresh;

7 usages   5 inheritors
public abstract class QualityChangingItem {
    /*Common operation such as increment quality, decrement quality, decrement sellin*/
    /*Abstract method updateQuality() which has to be defined by the extending class of individual item types*/
    18 usages
    final Item item;
    //The Quality value range is defined using QUALITY_MIN and QUALITY_MAX, and it provides ease to update the range in future.
    1 usage
    public static final int QUALITY_MIN = 0, QUALITY_MAX = 50;

    5 usages
    public QualityChangingItem(Item item) { this.item = item; }

    5 usages
    void incrementQuality() {
        if (item.quality < QUALITY_MAX) {
            item.quality = item.quality + 1;
        }
    }

    6 usages
    void decrementQuality() {
        if (item.quality > QUALITY_MIN) {
            item.quality = item.quality - 1;
        }
    }
```



```java
    5 usages
    public QualityChangingItem(Item item) { this.item = item; }

    5 usages
    void incrementQuality() {
        if (item.quality < QUALITY_MAX) {
            item.quality = item.quality + 1;
        }
    }

    6 usages
    void decrementQuality() {
        if (item.quality > QUALITY_MIN) {
            item.quality = item.quality - 1;
        }
    }

    3 usages
    void decrementSellIn() {
        item.sellIn = item.sellIn - 1;
    }

    1 usage   5 implementations
    abstract void updateQuality();
}
```

AgedBrieItem Class:

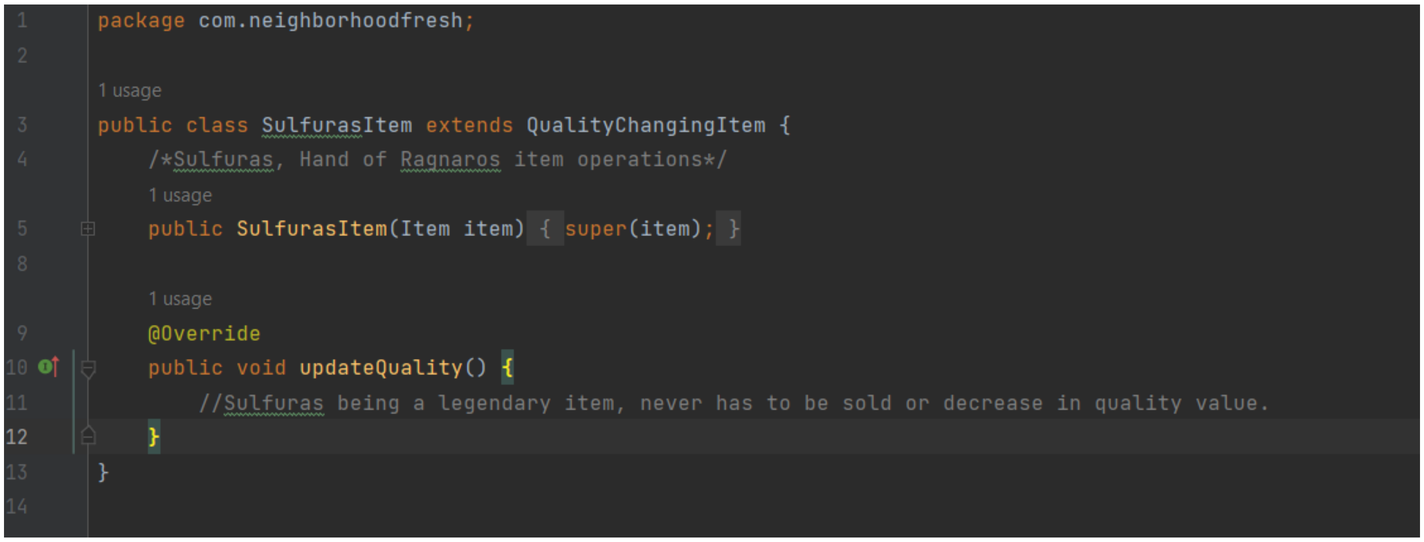
- Operations for the Item of Type – “Aged Brie”



```java
package com.neighborhoodfresh;

1 usage
public class AgedBrieItem extends QualityChangingItem {
    /*Aged Brie item operations*/
    1 usage
    public AgedBrieItem(Item item) {
        super(item);
    }

    1 usage
    @Override
    public void updateQuality() {
        incrementQuality();
        decrementSellIn();
        if (item.sellIn < 0) {
            incrementQuality();
        }
    }
}
```

SulfurasItem Class:

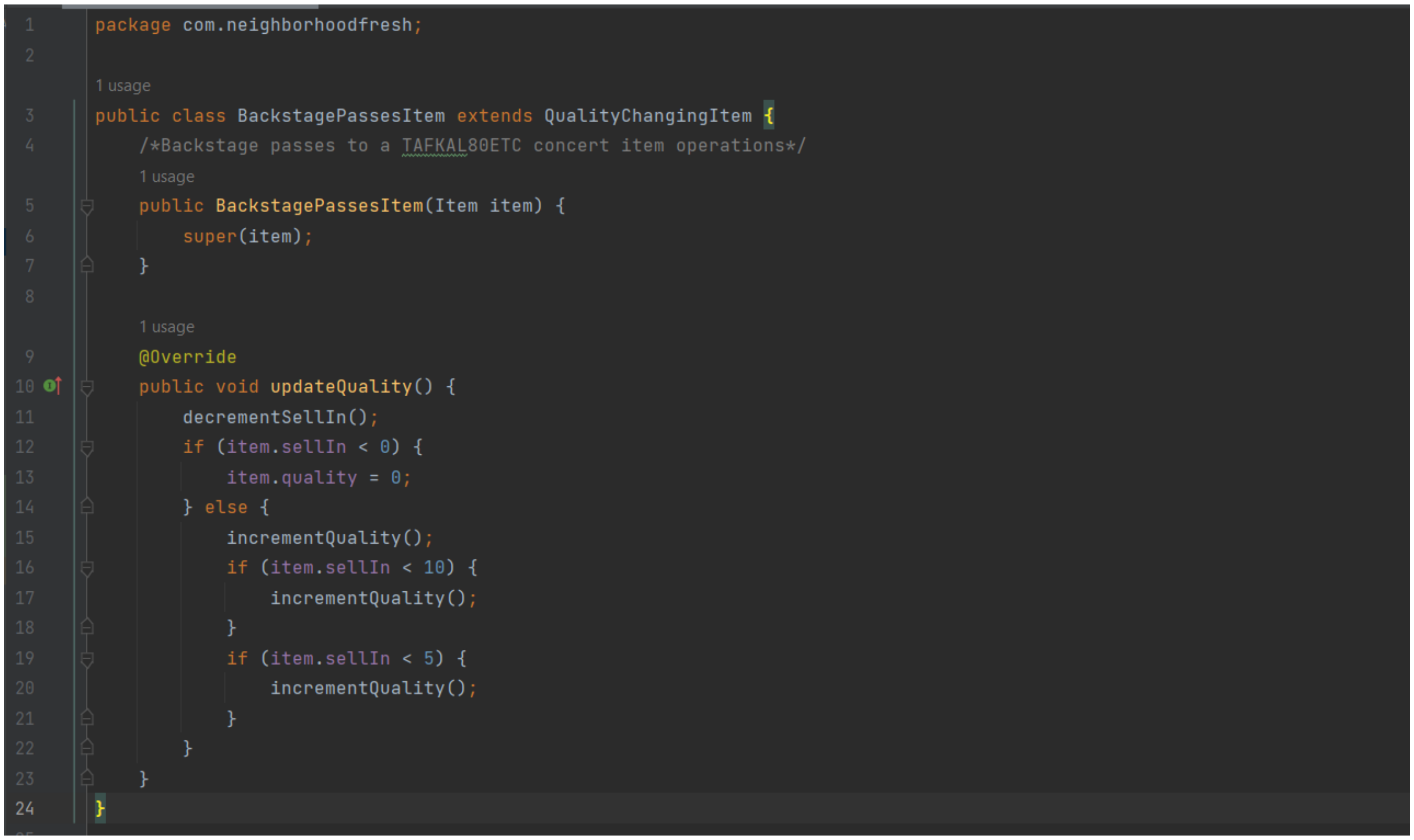
- Operations for the Item of Type – “Sulfuras, Hand of Ragnaros”



```
package com.neighborhoodfresh;

1 usage
public class SulfurasItem extends QualityChangingItem {
    /*Sulfuras, Hand of Ragnaros item operations*/
    1 usage
    public SulfurasItem(Item item) { super(item); }

    1 usage
    @Override
    public void updateQuality() {
        //Sulfuras being a legendary item, never has to be sold or decrease in quality value.
    }
}
```

BackstagePassesItem Class:

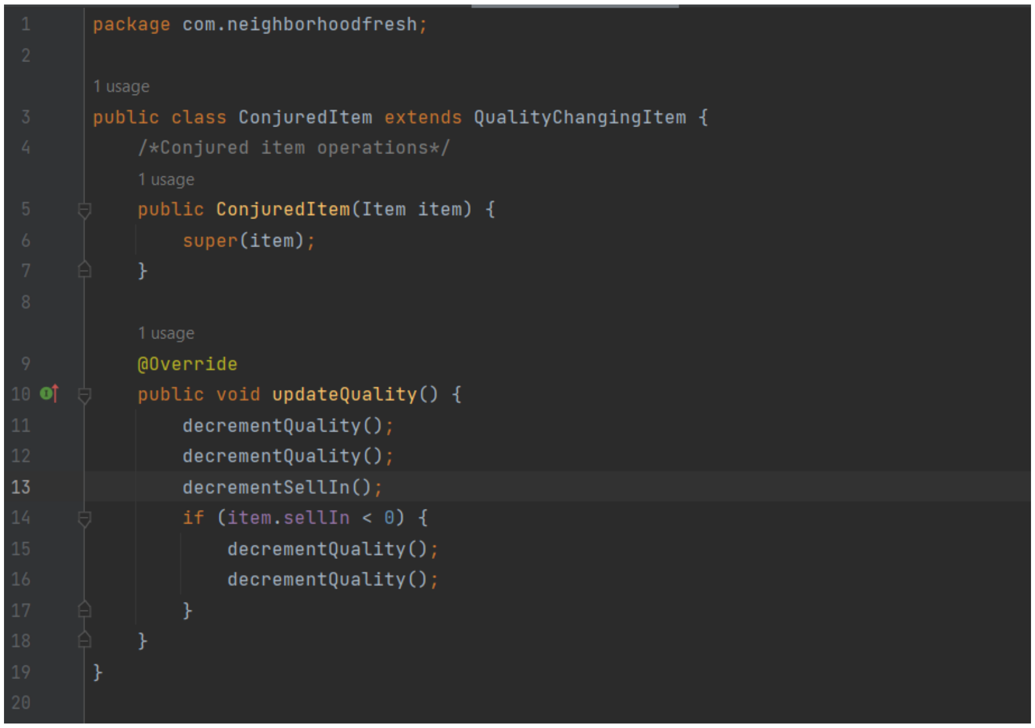
- Operations for Item of Type – “Backstage passes to a TAFKAL80ETC concert”



```
package com.neighborhoodfresh;

1 usage
public class BackstagePassesItem extends QualityChangingItem {
    /*Backstage passes to a TAFKAL80ETC concert item operations*/
    1 usage
    public BackstagePassesItem(Item item) {
        super(item);
    }

    1 usage
    @Override
    public void updateQuality() {
        decrementSellIn();
        if (item.sellIn < 0) {
            item.quality = 0;
        } else {
            incrementQuality();
            if (item.sellIn < 10) {
                incrementQuality();
            }
            if (item.sellIn < 5) {
                incrementQuality();
            }
        }
    }
}
```

ConjuredItem Class:

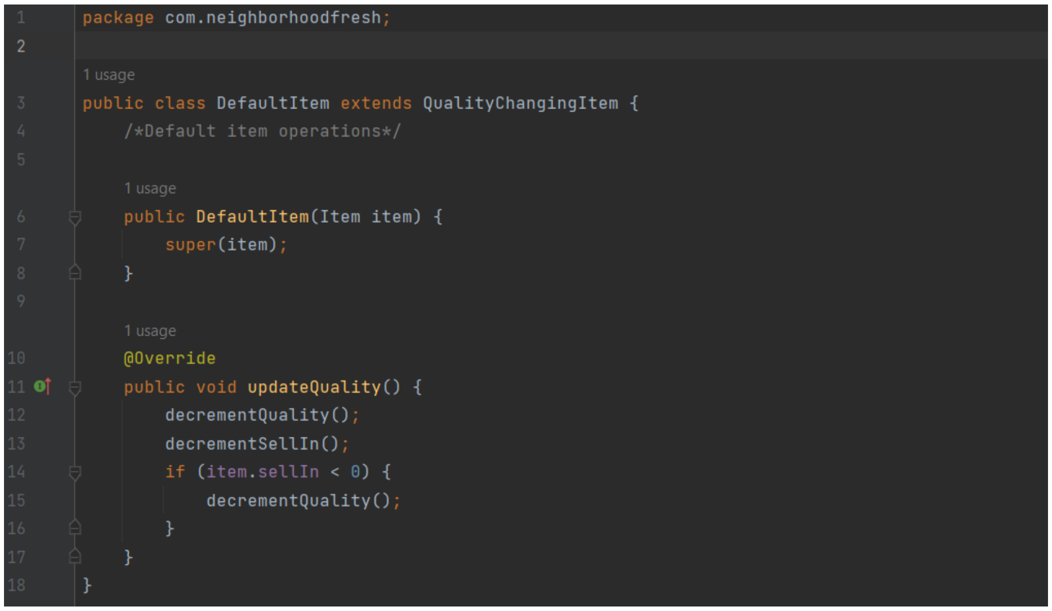
- Operations for Item of Type – “Conjured”
- “Conjured” item degrade in Quality twice as fast as normal items:
    - Therefore, the quality value decreases by 2 till sellin value reaches 0 and decreases by 4 after sellin value crosses 0 and goes negative, and stop when the quality value reaches 0.



```java
package com.neighborhoodfresh;

1 usage
public class ConjuredItem extends QualityChangingItem {
    /*Conjured item operations*/
    1 usage
    public ConjuredItem(Item item) {
        super(item);
    }

    1 usage
    @Override
    public void updateQuality() {
        decrementQuality();
        decrementQuality();
        decrementSellIn();
        if (item.sellIn < 0) {
            decrementQuality();
            decrementQuality();
        }
    }
}
```

DefaultItem Class:

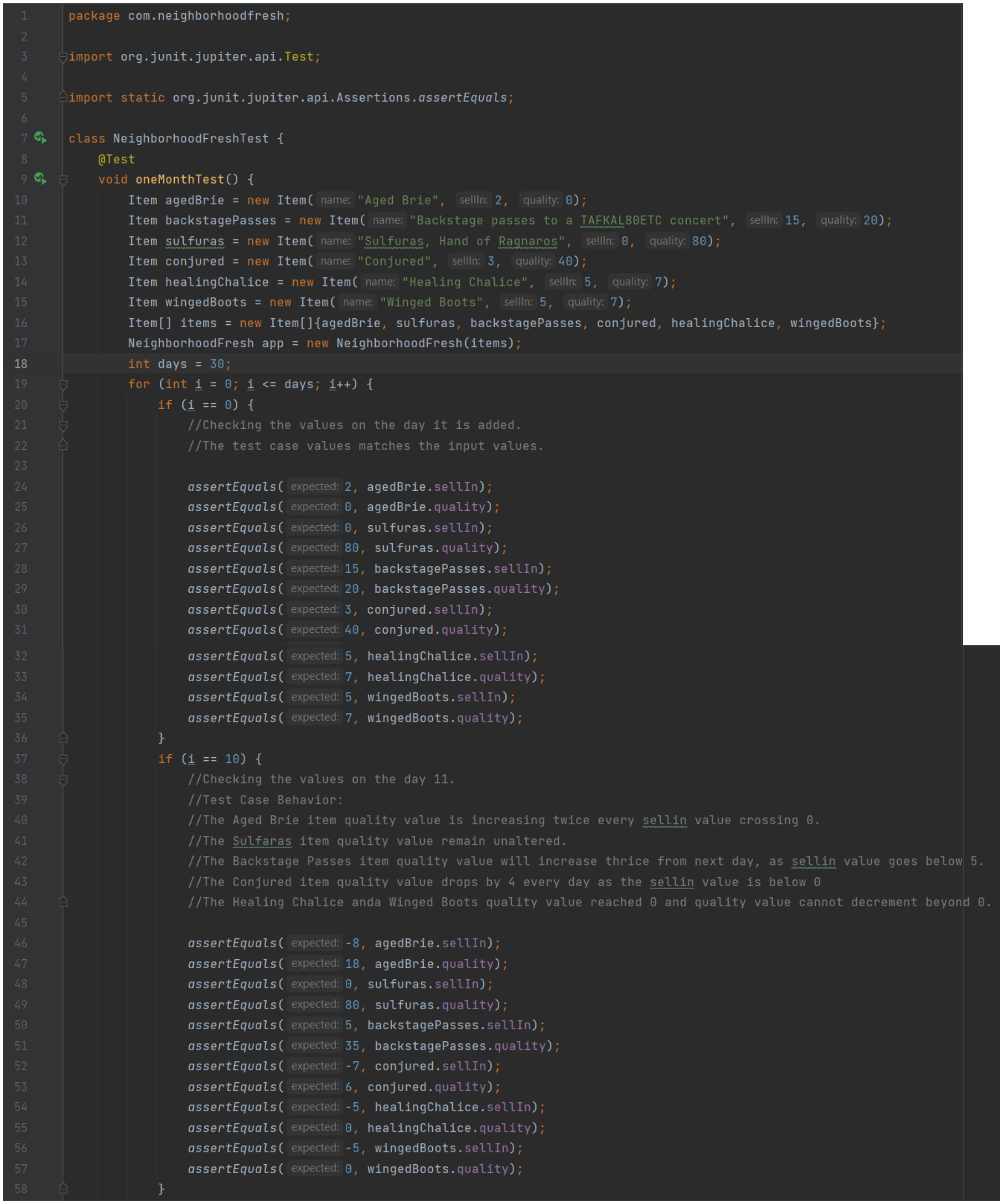
- Opertations for Items that does not belong to any of above mentioned item type.



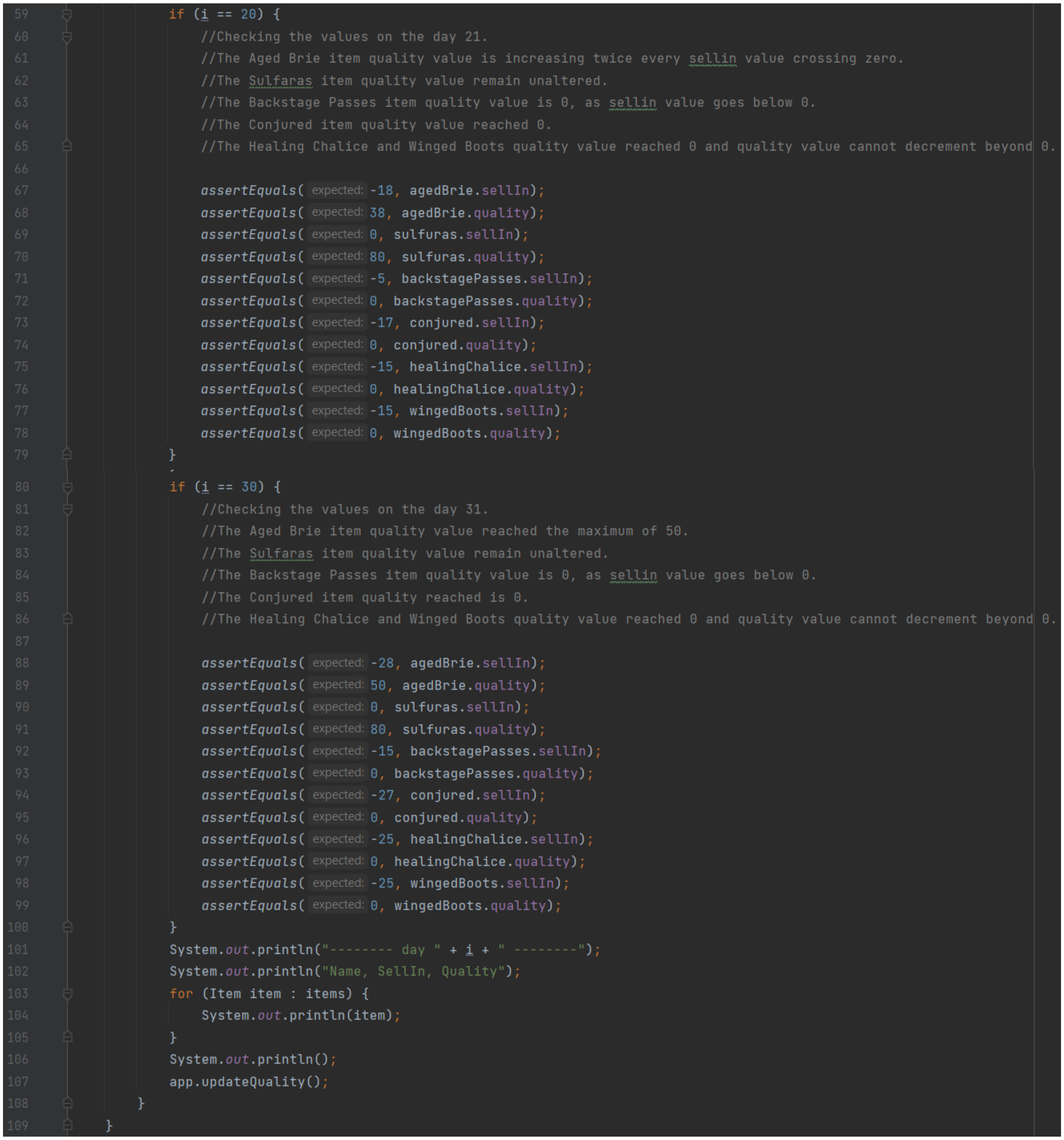
```java
package com.neighborhoodfresh;

1 usage
public class DefaultItem extends QualityChangingItem {
    /*Default item operations*/

    1 usage
    public DefaultItem(Item item) {
        super(item);
    }

    1 usage
    @Override
    public void updateQuality() {
        decrementQuality();
        decrementSellIn();
        if (item.sellIn < 0) {
            decrementQuality();
        }
    }
}
```

Test case:

- Test case with values for each item type and Junit test performed for every 10 days.

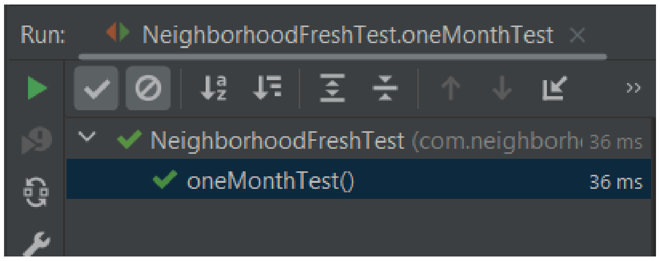


```java
package com.neighborhoodfresh;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

class NeighborhoodFreshTest {
    @Test
    void oneMonthTest() {
        Item agedBrie = new Item( name: "Aged Brie",  sellIn: 2,  quality: 0);
        Item backstagePasses = new Item( name: "Backstage passes to a TAFKAL80ETC concert",  sellIn: 15,  quality: 20);
        Item sulfuras = new Item( name: "Sulfuras, Hand of Ragnaros",  sellIn: 0,  quality: 80);
        Item conjured = new Item( name: "Conjured",  sellIn: 3,  quality: 40);
        Item healingChalice = new Item( name: "Healing Chalice",  sellIn: 5,  quality: 7);
        Item wingedBoots = new Item( name: "Winged Boots",  sellIn: 5,  quality: 7);
        Item[] items = new Item[]{agedBrie, sulfuras, backstagePasses, conjured, healingChalice, wingedBoots};
        NeighborhoodFresh app = new NeighborhoodFresh(items);
        int days = 30;
        for (int i = 0; i <= days; i++) {
            if (i == 0) {
                //Checking the values on the day it is added.
                //The test case values matches the input values.

                assertEquals( expected: 2, agedBrie.sellIn);
                assertEquals( expected: 0, agedBrie.quality);
                assertEquals( expected: 0, sulfuras.sellIn);
                assertEquals( expected: 80, sulfuras.quality);
                assertEquals( expected: 15, backstagePasses.sellIn);
                assertEquals( expected: 20, backstagePasses.quality);
                assertEquals( expected: 3, conjured.sellIn);
                assertEquals( expected: 40, conjured.quality);
                assertEquals( expected: 5, healingChalice.sellIn);
                assertEquals( expected: 7, healingChalice.quality);
                assertEquals( expected: 5, wingedBoots.sellIn);
                assertEquals( expected: 7, wingedBoots.quality);
            }
            if (i == 10) {
                //Checking the values on the day 11.
                //Test Case Behavior:
                //The Aged Brie item quality value is increasing twice every sellin value crossing 0.
                //The Sulfaras item quality value remain unaltered.
                //The Backstage Passes item quality value will increase thrice from next day, as sellin value goes below 5.
                //The Conjured item quality value drops by 4 every day as the sellin value is below 0
                //The Healing Chalice anda Winged Boots quality value reached 0 and quality value cannot decrement beyond 0.

                assertEquals( expected: -8, agedBrie.sellIn);
                assertEquals( expected: 18, agedBrie.quality);
                assertEquals( expected: 0, sulfuras.sellIn);
                assertEquals( expected: 80, sulfuras.quality);
                assertEquals( expected: 5, backstagePasses.sellIn);
                assertEquals( expected: 35, backstagePasses.quality);
                assertEquals( expected: -7, conjured.sellIn);
                assertEquals( expected: 6, conjured.quality);
                assertEquals( expected: -5, healingChalice.sellIn);
                assertEquals( expected: 0, healingChalice.quality);
                assertEquals( expected: -5, wingedBoots.sellIn);
                assertEquals( expected: 0, wingedBoots.quality);
            }
```



```java
        if (i == 20) {
            //Checking the values on the day 21.
            //The Aged Brie item quality value is increasing twice every sellin value crossing zero.
            //The Sulfaras item quality value remain unaltered.
            //The Backstage Passes item quality value is 0, as sellin value goes below 0.
            //The Conjured item quality value reached 0.
            //The Healing Chalice and Winged Boots quality value reached 0 and quality value cannot decrement beyond 0.

            assertEquals( expected: -18, agedBrie.sellIn);
            assertEquals( expected: 38, agedBrie.quality);
            assertEquals( expected: 0, sulfuras.sellIn);
            assertEquals( expected: 80, sulfuras.quality);
            assertEquals( expected: -5, backstagePasses.sellIn);
            assertEquals( expected: 0, backstagePasses.quality);
            assertEquals( expected: -17, conjured.sellIn);
            assertEquals( expected: 0, conjured.quality);
            assertEquals( expected: -15, healingChalice.sellIn);
            assertEquals( expected: 0, healingChalice.quality);
            assertEquals( expected: -15, wingedBoots.sellIn);
            assertEquals( expected: 0, wingedBoots.quality);
        }
        if (i == 30) {
            //Checking the values on the day 31.
            //The Aged Brie item quality value reached the maximum of 50.
            //The Sulfaras item quality value remain unaltered.
            //The Backstage Passes item quality value is 0, as sellin value goes below 0.
            //The Conjured item quality reached is 0.
            //The Healing Chalice and Winged Boots quality value reached 0 and quality value cannot decrement beyond 0.

            assertEquals( expected: -28, agedBrie.sellIn);
            assertEquals( expected: 50, agedBrie.quality);
            assertEquals( expected: 0, sulfuras.sellIn);
            assertEquals( expected: 80, sulfuras.quality);
            assertEquals( expected: -15, backstagePasses.sellIn);
            assertEquals( expected: 0, backstagePasses.quality);
            assertEquals( expected: -27, conjured.sellIn);
            assertEquals( expected: 0, conjured.quality);
            assertEquals( expected: -25, healingChalice.sellIn);
            assertEquals( expected: 0, healingChalice.quality);
            assertEquals( expected: -25, wingedBoots.sellIn);
            assertEquals( expected: 0, wingedBoots.quality);
        }
        System.out.println("-------- day " + i + " --------");
        System.out.println("Name, SellIn, Quality");
        for (Item item : items) {
            System.out.println(item);
        }
        System.out.println();
        app.updateQuality();
    }
}
```

Test case: **Pass,** output file (named) attached with the submission.



```
Run:   NeighborhoodFreshTest.oneMonthTest ×
✓ NeighborhoodFreshTest (com.neighborh 36 ms
    ✓ oneMonthTest()                  36 ms
```