## Hi and welcome to Cmoffix.

*Cmoffix* buys and sells only the finest goods in Berlin-like prices.

We have a system in place that updates our inventory for us.

* All items have a **SellIn** value which denotes the **number of days left 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 **updates both values** for every item.

These are the items currently in the system (keep in mind that more can be added in the future):

* T-Shirt
* Beer
* Scotch Bottle
* Basketball

Pretty simple, right? Well this is where it gets interesting:

* The **Quality** of an item **decreases by 1** every day (except for special cases described below)
* Once the intended sell date passed (meaning **SellIn value < 0**) , the item’s **Quality degrades twice as fast**
* The **Quality** of an item is never negative
* The **Quality** of an item is never more than 50
* And, there are these special rules (and more special rules can be added in the future)
  + The **Quality** of “Scotch bottle” increases (not decreases) by 1 every day
  + “Basketball”, being a legendary item, never has to be sold or decreases in **Quality**

**Requirements:**

You will get a partly working code written in a Java Spring-Boot, built in Gradle.

If you don’t know Gradle (a build tool similar to Maven) don’t worry about it. We will assist you.

Please complete the tasks below while making sure to document any code changes, so we can review it together.

You are free to work locally or to commit the code to a Git repository of your choice. Whatever you feel comfortable with.

**Task 1**

Download the project’s zip file from this link (Download as zip).

**Task 2**

Open the Project using your preferred IDE

**Task 3**

Try running the application using **ItemApplication** class.

Notes:

* You will get a failure. Fix it and run the application again.
* Make sure the application can run without any problem and you get a message containing the items available for Sell.
* This is the message that should appear at the bottom of the Console output in your IDE:

|  |
| --- |
| Current items: [Item{id=1, name='T-Shirt', sellIn=10, quality=20}, Item{id=2, name='Scotch Bottle', sellIn=2, quality=0}, Item{id=3, name='Beer', sellIn=5, quality=7}, Item{id=4, name='Basketball', sellIn=0, quality=50}] |

**Task 4**

Add a new **Rest Controller** to the application which should provide the following functionality:

* **findAll** – Returns a list of all available items
* **findById** - Gets an ID as parameter and returns the relevant Item
* **create** – Creates a new item

**Notes:**

* The controller should use the existing implementation of the ItemService.
* Note that you are not allowed to change the **Item** class.
* In order to **verify your controller is working correctly**, we’ve provided you a **Rest Test** class called **ItemRestIntegrationTest** (which must **NOT** be change)
* There might be additional bugs in the existing code you will need to fix.

**Task 5**

The method of calculating **SellIn** and **quality** (InMemoryItemService#updateQuality) is poorly written and therefore, difficult to maintain and update.

Please **refactor this code so that** it to be easy for maintenance and customization.

Keep in mind that it should be easy to:

* Add more items to the system
* Change each item’s calculation logic for **SellIn** and **Quality**

Notes:

* Keep in mind not to break existing unit-tests.
* Add more unit-tests if needed, to cover your refactoring.

**Task 6**

We have recently signed a supplier of **premium** items. This requires an update to our system: - "Premium" items change in Quality twice as fast as normal items. Premium is a category; it can be applied to any item to make it premium

For example:

* **Premium** Beer’s quality decrease by 2 every day, vs. non-premium Beer which decreases by 1 every day.
* **Premium** Scotch bottle quality – This modifies the special quality rule of "Scotch bottle" and sets its increase rate to twice as fast.

**You are required to add support in the premium feature described above.**

Start with just one item, say "Beer", and move on to other items if you have the time.

* Suggest a persistency model for the new premium feature.
  + Keep in mind that there might be more features in the future that will affect Quality calculations.
* **Make the needed code changes to support the “premium” feature for item’s “tagged” as premium.**
* Add **unit test(s)** for the "Premium Beer" (in InMemoryItemServiceTest class)
* Make sure to run all existing unit-tests in InMemoryItemServiceTest class to make sure current functionality is working correctly.

You should also add more tests to the class which verify the premium functionality is working as expected.

**Remember**! Your code should be expandable so you can easily add the premium ability to all the other items.

**Final Notes:**

In your refactoring considerations keep in mind the code should be easy to maintain and expand to more items and features which can determine how the quality and Sell-In values are calculated for each item type.

You are free to make any code changes **except for changing:**

1. The **Item** class
2. The **ItemRestIntegrationTest** class (for testing the rest controller you need to write in Task 4)

Good luck!