Assignment 1: Refactoring

Leitner, David - `leitnerd@technikum-wien.at`
Kreuzriegler, Matthias - `kreuzrie@technikum-wien.at`

1 Introduction

Working with legacy and hard to maintain code is one of the things we have to tackle as Software Engineers nearly on a daily basis. Thus, the goal of this assignment is to practice refactoring a very small, but still tricky code base into a more readable and maintainable solution. The example is based on a Coding Kata form Emily Bache which is called the "GildedRose-Refactoring-Kata". The goal is to refactor in small steps by applying the learned principles and concepts.

2 Tasks

1. Clone or download the GildedRose-Codebase from:
   https://github.com/emilybache/GildedRose-Refactoring-Kata.git
2. Understand the business requirements which are defined on the next page of this document.
3. Select the folder which contains the codebase in your preferred language. You can find a list of languages, plus the ignorable parts of the application at the end of the document.
4. Do *your* refactoring. Feel free to add your own tests.
5. Zip *only* the sub-folder, which contains your refactored codebase in the language you have chosen and upload it to moodle. You do *not* have to provide us any *GIT*-history.
6. In the provided field within moodle explain the strategy and principles you followed to refactor the codebase and why you think that those changes caused an improvement. Try to confirm this hypotheses with one of the learned metrics.

Please bear in mind, that the goal is not to rewrite the whole codebase, but to think explicitly about what can be improved and how to get there in babysteps.

3 Business Requirements

Hi and welcome to team Gilded Rose. As you know, we are a small inn with a prime location in a prominent city. 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. It was developed by someone, who has moved on to new adventures.

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 its 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.

Keep in mind, to make any changes to the update-quality method and add any new code as long as everything still works correctly. However, do not alter the Item class or Items property.

Language:	**Directory:**	**Files to change:**	**Ignorable*:**
C#	/csharp	GildedRose.cs GildedRoseTest.cs Item.cs	Program.cs ApprovalTest.cs
F#	/fsharp	GildedRose.fs GildedRoseTest.fs	"ApprovalTest" in the GildedRoseTest.fs "main" function in the GildedRose.fs
Java	/Java	GildedRose.java GildedRoseTest.java Item.java	TexttestFixture.java
Kotlin	/Kotlin	GildedRose.kt GildedRoseTest.kt Item.kt	TexttestFixture.kt
Javascript	/js-jasmine	gilded_rose.js gilded_rose_spec.js	-
Typescript	/TypeScript	gilded-rose.ts gilded-rose.spec.ts	golden-master-text-test.ts
Phyton	/python	gilded_rose.py test_gilded_rose.py	texttest_fixture.py

** this kata is often used to practice the concept of approval-testing. You can ignore any kind of approval-, text-file- or golden-master-test.*