M3105 - TP n°2 : Gilded Rose

Cet énoncé est disponible en ligne sur :
https://github.com/iblasquez/enseignement-iut-m3105

Bonjour et bienvenue dans l'équipe de la Rose dorée !

Comme vous le savez, notre petite taverne située à proximité d'une cité importante est dirigée par l'amicale aubergiste Allison.

Nous achetons et vendons uniquement les meilleurs produits. Malheureusement, la qualité de nos marchandises se dégrade constamment à l'approche de leur date de péremption.

Un système a été mis en place pour mettre à jour notre inventaire. Il a été développé par Leeroy, une personne pleine de bon sens qui est parti pour de nouvelles aventures.

Votre mission est d'ajouter une nouvelle fonctionnalité à notre système pour que nous puissions commencer à vendre un nouveau type de produit.

Mais d'abord, laissez-moi vous présenter notre système :

- Tous les éléments ont une valeur `sellIn` qui désigne le nombre de jours restant pour vendre l'article.
- Tous les articles ont une valeur `quality` qui dénote combien l'article est précieux.
- A la fin de chaque journée, notre système diminue ces deux valeurs pour chaque produit.

Plutôt simple, non ?

Attendez, ça devient intéressant :

- Une fois que la date de péremption est passée, la qualité se dégrade deux fois plus rapidement.
- La qualité (`quality`) d'un produit ne peut jamais être négative.
- "Aged Brie" augmente sa qualité (`quality`) plus le temps passe.
- La qualité d'un produit n'est jamais de plus de 50.
- "Sulfuras", étant un objet légendaire, n'a pas de date de péremption et ne perd jamais en qualité (`quality`)
- "Backstage passes", comme le "Aged Brie", augmente sa qualité (`quality`) plus le temps passe (`sellIn`) ; La qualité augmente de 2 quand il reste 10 jours ou moins et de 3 quand il reste 5 jours ou moins, mais la qualité tombe à 0 après le concert.

Nous avons récemment signé un partenariat avec un fournisseur de produit invoqué ("Conjured"). Cela nécessite une mise à jour de notre système :

- les éléments "Conjured" voient leur qualité se dégrader de deux fois plus vite que les objets normaux.

Vous pouvez faire les changements que vous voulez à la méthode `updateQuality` et ajouter autant de code que vous voulez, tant que tout fonctionne correctement. Cependant, nous devons vous prévenir, ne devez modifier en aucun cas la classe `Item` ou ses propriétés car cette classe appartient au gobelin de l'étage et il rentrera dans du rage instantanée et vous tuera sans délai : il ne croit pas dans le partage du code. (Vous pouvez ajouter une

méthode `updateQuality` et des propriétés statiques dans la classe `Item` si vous voulez, nous vous couvrirons)
Juste une précision, un produit ne peut jamais voir sa qualité augmenter au-dessus de 50, cependant "Sulfuras" est un objet légendaire et comme tel sa qualité est de 80 et il ne change jamais.

Voici l'énoncé du kata Gilded Rose, un grand classique pour s'entraîner au refactoring… 😉
Ce kata a été initialement créé en C# par Terry Hughes. Emily Bache l'a traduit dans de nombreux langages de programmation. Les spécifications (énoncées en français ci-dessus) ont également été traduites. Le tout est disponible sur le dêpôt github emilybache/GildedRose-Refactoring-Kata.

1. Mise en place du projet

Dans votre IDE préféré, créez un projet Maven `gildedRose` dans lequel vous y importerez **les sources** disponible dans le répertoire **ressources** `gildedrose` de ce dépôt : https://github.com/iblasquez/enseignement-iut-m3105-conception-avancee.

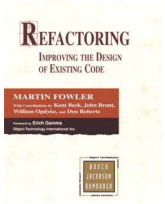
Faites en sorte que ce code compile !

Le formatage de ce code ne convient peut-être pas à certains d'entre vous car il ne correspond pas à la *forme* des programmes que vous avez l'habitude d'écrire avec votre IDE préféré (en terme d'accolade, d'indentation…)…
Pas de problème, commencez donc pas reformater ce code de manière à ce qu'il corresponde mieux à nos standards habituels (sous Eclipse sélectionnez **Source->Format**)

Faites en sorte que le code compile toujours !
(normalement le formatage n'a rien changé et le code doit continuer à compiler !)

Mettez votre projet sous git et procédez à votre premier commit avec un message du genre : « Commit initial : récupération du code legacy ».

Et maintenant, par où commencer ?



Martin Fowler dans Refactoring, Improving the Design of Existing Code propose de suivre la règle suivante :

When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature.

Avant d'ajouter à votre programme une nouvelle fonctionnalité, vous devez donc procéder à une phase de refactoring pour simplifier le code dont vous venez d'hériter et le rendre plus facile à maintenir (vous vous limiterez d'ailleurs dans ce TP uniquement à la seule phase de refactoring…)

Et pour garantir le comportement du système, il est indispensable, avant de procéder à un quelconque refactoring, de disposer d'un ensemble de tests automatisés. Ces tests serviront de *filet de sécurité* et permettront de remanier le code en toute sécurité : ce sont des tests de régression qui, pouvant être exécutés à tout moment, préviennent la non-régression du logiciel.

2. Mise en place des tests sur le projet

Consultez le fichier `InnTest`. Que contient-il ?
Rien,… Aucun test n'a été écrit par Leeroy, le développeur ayant écrit ce code 😉
La première chose à faire est donc d'écrire des tests et de les faire passer AU VERT sur ce code ***legacy*** !

A vous de jouer !!! …
Avant de procéder à un quelconque refactoring,
vous devez donc écrire tous les tests nécessaire pour couvrir 100% du code

Remarque: Pour la couverture de code, utilisez votre IDE (EclEmma avec Eclipse par exemple)

Une fois votre code couvert, commitez avec un message du genre :
« Mise en place du harnais de tests sur le code legacy ».

3. Refactoring let's go !!!

Si vous êtes arrivés ici, c'est que vous avez couvert 100% du code. Félicitations !

Vous allez donc maintenant pouvoir refactorer ce code en toute sécurité !!!
A vous de jouer !!!

Remarque: Pour un feedback rapide de tests en continu, n'hésitez pas à utiliser le plug-in Infinitest pour que vos tests soient automatiquement lancés dès qu'un changement est détecté dans le code source (c-a-d à chaque fois qu'un fichier modifié est sauvegardé).