# Re factoring Gilded Roses

## Equipe – binôme

ANNE Rudy

YAMEOGO Abdoul Moubarak

## Contexte

Il s’agit d’un projet de refactoring d’un code legacy (un « Code qui n’a pas de tests » : Michael Feathers). Le projet est proposé par Emily Bache sur GitHub, dans de langages différents. Pour notre cas, nous avons décidé de faire le refacto en JavaScript avec Jest pour l’intégration continue.

Vous trouverez le projet dans différents langages ici : <https://github.com/emilybache/GildedRose-Refactoring-Kata>

## Liste améliorations possibles par ordre de priorité :

* Ajouter un filet de tests qui enregistre le comportement de l’application
* Faire le refactoring du code :
  + Remplacer les occurrences par des classes et fonctions communes
  + Réorganiser la méthode updateQuality() qui est trop longue
  + Mettre en place un seul test unitaire qui fonctionne
* Retirer le filet de tests puis ajouter des tests unitaires

## Explications

Pour refactoriser ce code sans douleur, nous avons utilisé le principe du Golden Master :

1. Filet de test

En premier lieu, nous sommes aller au plus simple en modifiant le test dans le fichier gilded\_rose.test.js comme suite.

L’idée c’est de stocker les données obtenues lors l’appel du test dans un fichier (snapshots).

Dans un second temps, nous avons récupérez toutes les données et paramètres stockés afin de créer une nouvelle méthode de test.

Enfin, nous avons comparé les résultats afin d’obtenir la certification que les résultats sont identiques.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

1. Refactoring
   * Remplacer les occurrences par des classes et fonctions communes

Nous avons créé une classe pour chaque élément ou Item ( SulfurasItem, BackstageItem, ElixirItem, AgedBrieItem, DexterityVestItem) qui hérite la classe Item(). Cela nous a permis de repartir tout le code de la méthode updateQuality() de la classe Shop() dans les classes filles.

* + Réorganiser la méthode updateQuality() qui est trop longue

Pour plus de cohérence, nous avons rénommé la méthode updateQuality() en updateItem(). Cette dernière parcourt la liste des items passés en paramètres puis fait appel à la méthode updateItem() de chaque item. La classe mère Item() a aussi une méthode updateItem() qui fait appel à une fonction commune, extrait de l’ancienne methode updateQuality(), à quelques classes filles.

Ainsi les classes filles qui auront besoin de plus/moins de fonctionnalités surchageront cette méthode updateItem().

* + Mettre en place un seul test unitaire qui fonctionne

Avant de conclure sur le refactoring, nous nous sommes rassurés d’avoir mis en place un test unitaire qui fonctionne.

1. Tests unitaires

Pour les tests unitaires, on s’est basé sur les items de test qui se trouvaient dans le fichier texttest\_fixture.js que nous avons réadapté. L’idée c’est de retirer toutes les données de tests qu’on avait générées au tout début. Ainsi nous avons pu mettre en place des tests unitaires sur des cas spécifiques. Appeler la méthode updateItem() sur une liste d’items puis anticiper la valeur de sa qualité (this.quality) par exemple.