Rapport TP1 Génie Logiciel

Kata : Gilded Rose

Binôme :

- Alycia Belkacem
- Dahbia Moussi

Dans ce premier TP, nous allons effectuer le célèbre kata "Gilded Rose", un petit logiciel de gestion de stocks de bière. Le code fourni est un exemple classique de "code spaghetti" avec des structures de contrôle imbriquées et de la répétition de logique. Voici les méthodes de refactoring qu'on a appliqué pour simplifier ce code et le rendre plus lisible :

1. Extraction de méthodes : extraction des parties du code répétitives dans des méthodes distinctes, ce qui le rend plus lisible et facilite la compréhension du comportement de chaque type d'élément (élément régulier, "Aged Brie", "Backstage Pass", etc.). De plus, le code est organisé de manière à ce que les conditions et les étapes de mise à jour de qualité et de "sellIn" soient claires et explicites.

2. Utilisation de constantes : Utilisation des constantes pour les valeurs magiques, déclarées en tant que champs statiques dans la classe GildedRose telles que les noms des produits ou la qualité maximale (50). Ce qui rendra le code plus lisible et facilitera les modifications futures.

3. Élimination de la duplication de code : Identification et élimination de la duplication de code. Par exemple, la logique de diminution de qualité est répétée à plusieurs endroits. Vous pouvez créer une méthode privée pour cela et l'appeler depuis les méthodes appropriées pour éliminer la duplication

4. Utilisation de structure de données et changement de l'architecture : Réorganisation du code pour qu'il soit encore plus modulaire et suive les principes SOLID. Le code utilise principalement des structures de contrôle conditionnelles (if-else) pour déterminer le comportement en fonction du type d'article. Cela fonctionne correctement pour ce petit nombre de types d'articles, mais si on a besoin de gérer un nombre croissant de types d'articles, il peut devenir difficile à maintenir. Une approche plus extensible consiste à utiliser des structures de données telles que des classes et la polymorphie. Pour cela on a créé des classes distinctes pour chaque type d'article (Regular, Aged Brie, Backstage Pass) et les faire implémenter une interface commune. Cela va permettre de déplacer la logique de mise à jour de chaque type d'article dans sa propre classe, facilitant ainsi l'ajout de nouveaux types d'articles à l'avenir.

5. Simplification de code : Utilisation des opérateurs ternaires pour réduire les instructions if en une seule ligne pour chaque condition. Cela rend le code plus concis tout en maintenant la même logique de mise à jour de la qualité.

Ces différentes étapes de refactoring ont permis l'amélioration de la qualité du code, en le rendant plus lisible, maintenable et évolutif tout en gardant le même comportement. Cependant, pendant le processus de refactoring, certaines mutations PIT ne sont plus couvertes, et cela est dû à plusieurs raisons :

1. **Suppression de code mort** : la suppression de code non utilisé ou qui était incorrect, certaines mutations qui ciblaient ce code ne sont plus pertinentes.
2. **Réduction de complexité** : L'élimination des conditions inutiles ou des branchements complexes, cela peut réduire le nombre de mutations nécessaires pour obtenir une bonne couverture de code.
3. **Réutilisation de code** : En regroupant du code répété, certaines mutations peuvent être éliminées, car la même logique est utilisée dans plusieurs endroits du code.
4. **Changements mineurs** : Certaines mutations peuvent être couvertes par d'autres mutations plus générales après le refactoring.

Le fait que certaines mutations ne soient plus couvertes n'est pas un problème en soi, tant que le comportement global du code reste inchangé, et que les tests passent avec succès. Cela peut même indiquer que le code est devenu plus clair et plus efficace.

En fin de compte, l'objectif principal du refactoring est d'améliorer la qualité du code sans introduire de nouvelles erreurs, et la couverture des mutations est un outil utile pour vérifier que le comportement global du code n'a pas été altéré.