Rapport de TP : Test de Couverture et Mutation de Code

I. Objectifs

- **Fonctionnalité Principale** : Développer un algorithme capable de générer l'anagramme d'un mot saisi par l'utilisateur, en utilisant le tri par insertion pour ordonner les caractères du mot en ordre alphabétique.
- **Technologies Utilisées** : Java, avec une utilisation intensive des expressions lambda pour la modularité des tests.

II. Spécification de l'Algorithme

L’algorithme développé répond aux spécifications suivantes :

1. **Entrée Utilisateur** : Récupération d'un mot saisi par l'utilisateur.
2. **Traitement** :
    - Conversion du mot en tableau de caractères.
    - Application du tri par insertion pour ordonner les caractères.
3. **Sortie** : Retour du mot trié, formant ainsi un anagramme par ordre alphabétique.

III. Implémentation de l'Algorithme

L'algorithme utilise le tri par insertion adapté aux tableaux de caractères en Java. Des expressions lambda sont employées pour passer différentes fonctions comme paramètres lors des tests, permettant une flexibilité accrue.

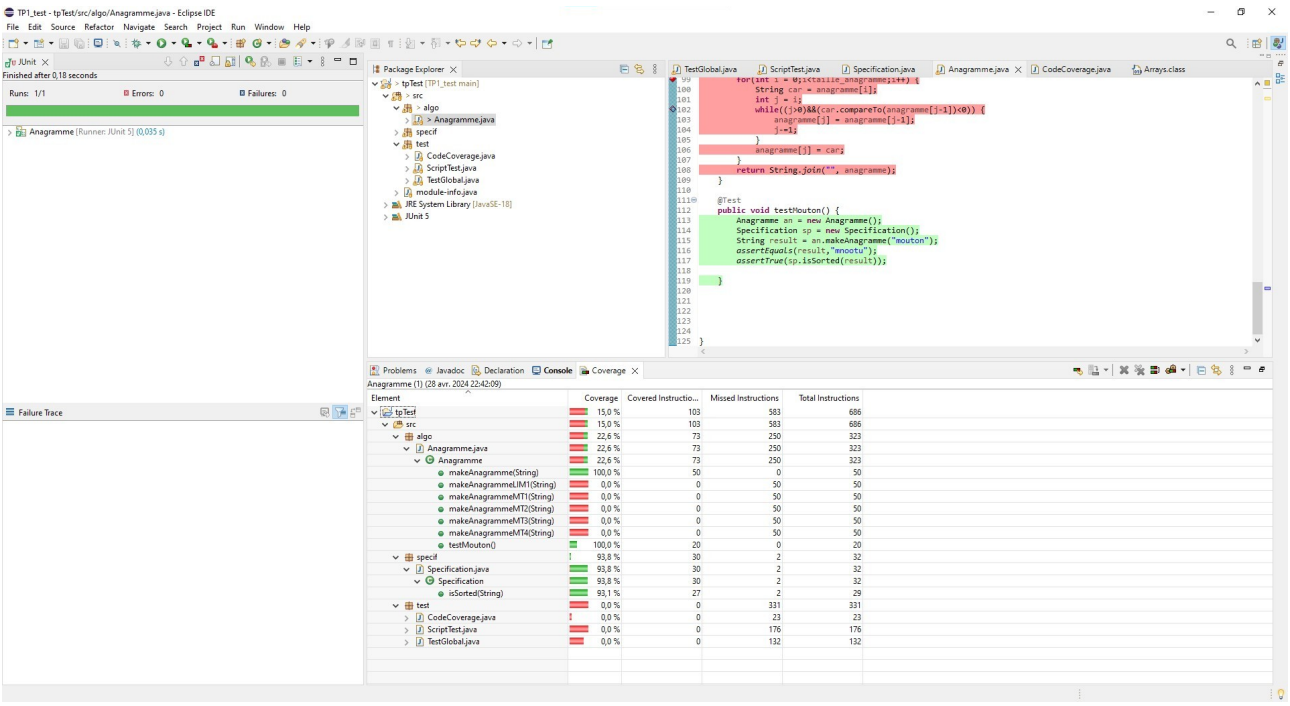
IV. Script de Test

Un script de test automatisé a été développé pour lire les jeux de tests et les oracles depuis un fichier. Ce script exécutable utilise Java pour :

- Exécuter les tests.
- Générer un rapport d'analyse indiquant les tests exécutés et leur résultat (passé ou échoué).

V. Couverture des Instructions

L'utilisation d'un outil de couverture de code déjà présent dans l'IDE Eclipse a permis de vérifier les instructions impliquées dans la génération de l'anagramme pour le mot "mouton",



VI. Test Fonctionnel

Une suite de tests a été créée en utilisant la méthode des partitions de classes d'équivalence et des tests aux limites. Cette suite est conçue pour être facilement extensible, permettant l'ajout de nouveaux cas de test sans modification substantielle du cadre de test existant.

VII. Test de Mutation

Quatre critères de mutation ont été définis et appliqués pour créer des mutants du programme de référence. Chaque mutant a ensuite été testé avec la suite de tests fonctionnels :

1. **Mutation 1** : Remplacement de < par <= dans une condition de boucle, causant un débordement.
2. **Mutation 2** : Remplacement de < par > dans une condition, inversant l'ordre du tri.
3. **Mutation 3** : Remplacement de > par <= dans une condition, entraînant également un débordement.
4. **Mutation 4** : Remplacement de - par + dans une opération de boucle, empêchant le tri correct.

VIII. Analyse des Résultats

L'analyse des tests de mutation a montré une capacité variée de la suite de tests à détecter des erreurs introduites. Des ajustements ont été nécessaires pour améliorer la détection des fautes par les tests, en particulier pour les mutations qui entraînent des changements subtils dans la logique de traitement.

IX. Conclusion

Le TP a permis de mettre en pratique des techniques avancées de test et de couverture de code en Java. Les tests de mutation ont particulièrement mis en évidence l'importance de concevoir une suite de tests robuste et exhaustive pour assurer la fiabilité du logiciel et l’utilisation des lambdas a grandement amélioré la modularité du code.