RAPPORT TÂCHE 2

Test : testFromFloat_WrapperDelegatesCorrectly

Intention :

Ce test a pour objectif de vérifier que la méthode fromFloat écrit correctement une valeur flottante dans un tableau d'octets lorsqu'aucun décalage n'est fourni (offset = 0). Il valide que le wrapper appelle correctement la méthode principale avec le décalage par défaut.

Données de test :

- Un tableau de 4 octets, correspondant à la taille standard d'un float.
- La valeur flottante 42.42f choisie pour représenter un nombre non trivial et s'assurer que l'écriture manipule correctement les bits.

Oracle :

Après écriture de la valeur dans le tableau, toFloat doit retourner exactement la même valeur flottante.

Test : testSubList_NormalRange

Intention :

Ce test vérifie que la méthode subList retourne correctement une sous-liste lorsqu'on lui fournit un intervalle correct. Il valide que les indices sont bien gérés et que l'extraction est correcte.

Données de test :

Liste d'entrée : [1, 2, 3, 4, 5], indices : (1,4). La liste est sortée pour faciliter la vérification, l'intervalle choisi permet de tester l'extraction de la sous-liste.

Oracle :

Le résultat attendu est [2, 3, 4]. Si la méthode retourne cette sous-liste, le test est réussi ; sinon.

Test : testCalcSortOrder_InvalidLength

Intention :

Vérifie que calcSortOrder lance une IllegalArgumentException lorsque la longueur fournie ne correspond pas à la taille des tableaux.

Données de test :

- Cas 1 : arr1=[1,2,3], arr2=[4,5,6], longueur = 4 -> longueur trop grande.
- Cas 2 : arr1=[1,2,3], arr2=[7,8], longueur = 3 -> arr2 trop court.

Oracle :

Dans les deux cas, une IllegalArgumentException doit être levée, confirmant la validation de la longueur des tableaux.

Test : testApplyOrder_InvalidOrderLength_ThrowsException

Intention :

Vérifie que applyOrder rejette un ordre dont la longueur est supérieure à celle du tableau source.

Données de test :

- Tableau : [10, 20, 30]
- Ordre fourni : [2, 1, 0, 3]

Oracle :

Une IllegalArgumentException doit être levée, validant que la méthode vérifie correctement la cohérence des longueurs.

Test : testZero

Intention :

Vérifie que la méthode zero(int) génère correctement un tableau rempli de zéros de la taille spécifiée.

Données de test :

Trois cas testés : taille 0, taille 1 et taille 5.

Oracle :

Pour chaque cas, le tableau généré doit correspondre exactement au tableau attendu rempli de zéros sans quoi une erreur est retournée.

Test : testRemoveConsecutiveDuplicatesErrors

Intention :

Vérifie que removeConsecutiveDuplicates lance une exception lorsque la longueur fournie est invalide.

Données de test :

- Tableau : [3, 3, 4, 2, 1, -3, -3, 9, 3, 6, 6, 7, 7]
- Longueurs testées : -1, arr.length + 1 (out of range)

Oracle :

- Longueur négative : IllegalArgumentException
- Longueur trop grande : ArrayIndexOutOfBoundsException

Test : testFromDouble_WrapperDelegatesCorrectly

Intention :

Ce test vérifie que la méthode fromDouble écrit correctement une valeur double dans le tableau d'octets, en utilisant un offset =0.

Données de test :

- Tableau d'octets de taille 8 (taille d'un double).
- Valeur double testée : 123456.789.

Oracle :

Après écriture, la lecture avec toDouble doit retourner exactement la même valeur.

Test : testFromFloat_WrapperDelegatesCorrectly

Intention:

Vérifier que la méthode fromFloat écrit correctement la valeur flottante dans le tableau d'octets en appelant la version principale avec un décalage de 0.

Données de test :

- Un tableau d'octets de taille 4.
- Une valeur flottante simple : 42.42f.

Oracle :

Après écriture, la lecture du tableau via toFloat(bytes, 0) doit retourner la même valeur que celle passée en paramètre.

Test de JavaFaker :

Nous utilisons javafaker pour randomiser les données de test dynamiquement pour augmenter la couverture de test de la fonction et éviter des données de test prédéfinies permettant de vérifier le vrai fonctionnement de la fonction

---

Test : testCalcSortOrder_IntArrayList_UnequalSize_ThrowsException_Faker

Intention :

Vérifie que calcSortOrder rejette deux listes de tailles différentes en lançant une IllegalArgumentException. L'utilisation de JavaFaker permet de générer dynamiquement des listes d'entiers aléatoires, évitant le hardcoding et augmentant la couverture des cas.

Données de test :

- Tailles des listes générées aléatoirement (3–10), en s'assurant qu'elles sont inégales.
- Contenu des listes : entiers aléatoires 1–100, mais seule la différence de taille importe pour le test.

Oracle :

Une IllegalArgumentException doit être levée, confirmant que la méthode valide la taille des listes, même avec des contenus aléatoires.

Pitest :

Pour générer les rapport de mutation, il faut cd à core puis exécuter cette commande : mvn test-compile org.pitest:pitest-maven:mutationCoverage en suite, le rapport sera présent dans core/target/pit-reports/index.html

Sans les nouveaux test :

Package Summary

com.graphhopper.util

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	88%	198/226	91%	230/252	97%	230/236

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
ArrayUtil.java	84%	92/109	86%	91/106	96%	91/95
BitUtil.java	91%	106/117	95%	139/146	99%	139/141

Avec les nouveaux test :

Pit Test Coverage Report

Package Summary

com.graphhopper.util

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	94%	213/226	94%	238/252	97%	238/245

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
ArrayUtil.java	94%	103/109	92%	97/106	95%	97/102
BitUtil.java	94%	110/117	97%	141/146	99%	141/143

Résultats Avant l'Ajout des Nouveaux Tests :

Métriques Globales (Package com.graphhopper.util) :

- Line Coverage : 88% (198/226 lignes)
- Mutation Coverage : 91% (230/252 mutants détectés)
- Test Strength : 97% (230/236)

ArrayUtil.java :

- Line Coverage : 84% (92/109 lignes)

- Mutation Coverage : 86% (91/106 mutants)
- Test Strength : 96% (91/95)

BitUtil.java :
- Line Coverage : 91% (106/117 lignes)
- Mutation Coverage : 95% (139/146 mutants)
- Test Strength : 99% (139/141)

Résultats Après l'Ajout des Nouveaux Tests :

Métriques Globales (Package com.graphhopper.util) :
- Line Coverage : 94% (213/226 lignes) - Amélioration de +6%
- Mutation Coverage : 94% (238/252 mutants détectés) - Amélioration de +3%
- Test Strength : 97% (238/245) - Stable

ArrayUtil.java :
- Line Coverage : 94% (103/109 lignes) - Amélioration de +10%
- Mutation Coverage : 92% (97/106 mutants) - Amélioration de +6%
- Test Strength : 95% (97/102) - Baisse de -1%

Les nouveaux tests ont permis de couvrir 11 lignes supplémentaires et de détecter 6 mutants additionnels dans ArrayUtil. Les tests ajoutés ciblent principalement les validations d'erreurs (exceptions) et les cas limites non testés auparavant.

BitUtil.java :
- Line Coverage : 94% (110/117 lignes) - Amélioration de +3%
- Mutation Coverage : 97% (141/146 mutants) - Amélioration de +2%
- Test Strength : 99% (141/143) - Stable

Les deux nouveaux tests pour BitUtil ont permis de couvrir 4 lignes supplémentaires et de détecter 2 mutants additionnels, correspondant aux méthodes wrapper fromFloat(byte[], float) et fromDouble(byte[], double) qui n'étaient pas explicitement testées.

<u>Mutants Détectés pour ArrayUtil:</u>

Mutant à la ligne 49 :
- replaced return value with null for ArrayUtil::zero Test qui le détecte : testZero()
- Raison de la détection : Ce mutant remplace la valeur de retour de la méthode zero(int) par null au lieu de retourner un IntArrayList rempli de zéros. Le test testZero() vérifie explicitement que la méthode retourne des tableaux corrects pour différentes tailles (0, 1, et 5). Lorsque le mutant retourne null, les assertions assertEquals() échouent immédiatement car on compare null à un IntArrayList attendu. Sans ce test, ce mutant n'avait aucune couverture (NO_COVERAGE), et grâce à testZero(), il passe à KILLED.

Mutants des lignes 233, 234, et 235 :
- Mutants liés à subList Test qui les détecte : testSubList()

- Raison de la détection : Ces mutants affectent la méthode subList(IntArrayList, int, int) qui extrait une sous-liste. Dans le premier rapport, ces 5 mutants étaient marqués NO_COVERAGE car aucun test n'appelait la méthode subList. Les mutations incluent : modification des conditions de boucle et des incréments, remplacement de la soustraction par une addition dans le calcul d'indice, suppression de l'appel à add, et remplacement de la valeur de retour par null. Le test testSubList() vérifie que l'extraction des indices (1, 4) de [1,2,3,4,5] retourne [2,3,4]. Chaque mutation produit un résultat incorrect qui vérifié par ce test, ce qui fait échouer cette assertion, permettant de détecter tous ces mutants qui passent de NO_COVERAGE à KILLED dans le nouveau rapport.

Mutants Détectés pour BitUtil:

Mutant à la ligne 43 :
- removed call to com/graphhopper/util/BitUtil::fromDouble Test qui le détecte : testFromDouble_WrapperDelegatesCorrectly()
- Raison de la détection : Ce mutant supprime l'appel à la méthode fromDouble(byte[], double, int) dans la version wrapper fromDouble(byte[], double). Le test écrit une valeur double (123456.789) dans un tableau d'octets via le wrapper, puis la relit avec toDouble(bytes, 0). Si l'appel à fromDouble est supprimé (mutant), le tableau reste vide ou non modifié, et la valeur relue ne correspond pas à la valeur écrite. L'assertion assertEquals(value, result, 0.0000001d) échoue, détectant ainsi le mutant qui passe de NO_COVERAGE à KILLED.

Mutant à la ligne 65 :
- removed call to com/graphhopper/util/BitUtil::fromFloat Test qui le détecte : testFromFloat_WrapperDelegatesCorrectly()
- Raison de la détection : De manière similaire au mutant de la ligne 43, ce mutant supprime l'appel à la méthode fromFloat(byte[], float, int) dans la version wrapper fromFloat(byte[], float). Le test écrit une valeur float (42.42f) dans un tableau d'octets via le wrapper, puis la relit avec toFloat(bytes, 0). Si l'appel est supprimé, le tableau n'est pas modifié correctement et la valeur relue diffère de la valeur attendue. L'assertion assertEquals(value, result, 0.000001f) échoue, permettant de détecter ce mutant qui passe de NO_COVERAGE à KILLED.

Conclusion Les nouveaux tests ont permis d'améliorer significativement les métriques de mutation : +8 mutants détectés au total (6 pour ArrayUtil et 2 pour BitUtil). La couverture de ligne est passée de 88% à 94% au niveau du package, démontrant l'efficacité des tests ajoutés pour couvrir les cas limites et les validations d'erreurs précédemment non testées.