Dossier de tests pour la calculatrice

1. Introduction

Objectifs du test

L'objectif de ces tests est de vérifier le bon fonctionnement des opérations de base implémentées dans la calculatrice, à savoir l'addition, la soustraction, la multiplication et la division. Ces tests permettent d'identifier d'éventuelles erreurs et de valider la robustesse des algorithmes utilisés.

Présentation de ce qui est testé

Les méthodes suivantes de la classe `Calculatrice` sont testées :

- `addition(int a, int b)`
- `soustraction(int a, int b)`
- `multiplication(int a, int b)`
- `division(int a, int b)`

2. Description du test

Type de test

Les tests effectués sont des **tests unitaires**, réalisés à l'aide de **JUnit**. Ils visent à tester chaque fonction indépendamment. Pour les tests standards (sans exception attendue), nous utilisons la méthode **assertTrue** qui vérifie que le résultat obtenu correspond à la valeur attendue. Pour les tests d'exception, nous utilisons la méthode **assertThrows** qui permet de vérifier qu'une exception spécifique est bien levée dans certaines conditions.

Stratégie adoptée

La stratégie adoptée est celle de la **boîte noire**, où nous vérifions les entrées et sorties sans nous préoccuper de l'implémentation interne. Nous avons également utilisé la **partition d'équivalence**, où nous testons différents cas représentatifs des entrées possibles.

Présentation de la procédure

Chaque fonction est soumise à des cas de test spécifiques, prenant en compte des valeurs positives, négatives et nulles. Pour les cas standards, nous vérifions avec assertTrue que le résultat de l'opération correspond à la valeur attendue. Pour la division, nous avons également testé les cas provoquant des exceptions, en utilisant assertThrows pour vérifier que les exceptions sont bien levées dans les cas problématiques (division par zéro ou diviseur négatif).

3. Tests

Cas de test et résultats

3.1 Addition

Cas de test	Entrée (a, b)	Attendu	Résultat
a et b positifs	(3, 2)	5	☑
a = 0, b > 0	(0, 4)	4	☑
a > 0, b = 0	(5, 0)	5	☑
a et b négatifs	(-4, -6)	-10	☑
a < 0, b > 0,	(-3, 5)	2	☑
a > 0, b < 0	(7, -7)	0	☑

3.2 Soustraction

Cas de test	Entrée (a, b)	Attendu	Résultat
a > 0, b > 0	(5, 2)	3	☑
a = 0, b > 0	(0, 4)	-4	☑
a > 0, b = 0	(3, 0)	3	☑
a et b négatifs	(-5, -2)	-3	☑
a > 0, b < 0	(4, -2)	6	☑

3.3 Multiplication

Cas de test	Entrée (a, b)	Attendu	Résultat
a > 0, b > 0	(3, 2)	6	☑
a = 0, b > 0	(0, 4)	0	☑
a et b négatifs	(-3, -2)	6	☑
a > 0, b < 0	(4, -2)	-8	☑

3.4 Division

Cas de test	Entrée (a, b)	Attendu	Résultat
a > 0, b > 0	(6, 2)	3	☑
a = 0, b > 0	(0, 4)	0	☑
a > 0, b = 0	(5, 0)	Exception	☑
a < 0, b < 0	(-6, -2)	Exception	☑
a = 0, b < 0	(0, -3)	Exception	☑
a < 0, b = 0	(-4, 0)	Exception	☑

Cas de test	Entrée (a, b)	Attendu	Résultat
a = 0, b = 0	(0, 0)	Exception	☑
a < 0, b > 0	(-6, 3)	-2	☑
a > 0, b < 0	(6, -2)	Exception	☑

Analyse des résultats

Tous les tests ont été validés avec succès. Les résultats obtenus sont conformes aux attentes. Pour les tests standards, nous avons utilisé assertTrue pour vérifier que chaque opération renvoie le résultat correct. Pour les cas d'erreur, la gestion est bien implémentée et testée en utilisant assertThrows pour vérifier que les exceptions sont bien levées dans les cas problématiques, notamment:

- Division par zéro
- Division avec un diviseur négatif

4. Conclusion

La calculatrice a passé l'ensemble des tests unitaires sans erreur. Elle est capable d'effectuer correctement les opérations de base tout en gérant les cas particuliers. L'utilisation des assertions JUnit modernes (assertTrue pour les tests standards et assertThrows pour les exceptions) permet de tester efficacement tous les scénarios, rendant le code de test plus lisible et plus maintenable.