**Rapport sur l'implémentation de la classe MatriceModulo – Labo05**

**Nathan Füllemann – Mathéo Lopez**

**Objectifs Pédagogiques**

1. **Conception Orientée Objet avec Encapsulation** : Assurer l'intégrité des données de chaque matrice à travers des attributs privés.
2. **Lancer des Exceptions pour la Validation des Paramètres** : Garantir que les opérations entre matrices sont valides en tenant compte du module et des dimensions.
3. **Modélisation d'Opérations Arithmétiques** : Factoriser les opérations entre matrices pour rendre l'extension du code aisée et réduire les dépendances.

**Hypothèses de Travail et Choix de Conception**

Pour ce projet, l’objectif principal était de représenter des matrices modulo n avec des capacités d'initialisation, d'affichage, et d'opérations arithmétiques. Bien que nous ayons envisagé l’utilisation d'interfaces pour découpler les opérations, nous n'avons pas utilisé cette approche car les interfaces et la gestion des structures de classes n’ont pas encore été abordées dans notre cours.

Nous avons donc choisi de regrouper les fonctionnalités principales dans la classe MatriceModulo et de factoriser le code des opérations arithmétiques de manière à ce qu’il soit facilement extensible.

**Conception et Modélisation UML**

La classe MatriceModulo est modélisée selon les principes suivants :

1. **Attributs Privés** :
   * int[][] matrice : Le tableau 2D représentant les éléments de la matrice.
   * int rows et int cols : Dimensions de la matrice.
   * int modulo : Valeur maximale pour chaque élément, correspondant au paramètre modulo n.
2. **Constructeurs** :
   * **Matrice Vide** : Création d'une matrice de dimensions données, initialisée à zéro.
   * **Matrice Aléatoire** : Génération aléatoire d'éléments compris entre 0 et n−1 si un drapeau booléen est passé.
   * **Matrice avec Valeurs Données** : Permet de définir une matrice en passant directement ses valeurs, en appliquant le modulo à chaque élément.
3. **Méthode d'Affichage** : Affiche le contenu de la matrice pour visualiser facilement les opérations.
4. **Méthode operationMatrice** :
   * La méthode générique operationMatrice prend une autre matrice MatriceModulo, ainsi qu’une chaîne de caractères spécifiant l'opération souhaitée.
   * Elle vérifie les dimensions et le module des matrices avant d'appliquer l'opération, en gérant les cas où les tailles de matrices diffèrent.
   * Elle gère trois opérations : addition, soustraction et multiplication composante par composante.

**Modèle UML**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**Une image contenant texte, écriture manuscrite, écrit à la main

Description générée automatiquementCode :**

**Une image contenant texte, écriture manuscrite, écrit à la main

Description générée automatiquement**

**Documentation des Tests**

**Programme de Test Principal**

Nous avons implémenté un programme de test dans la méthode main qui permet de créer et de tester deux matrices aléatoires de tailles différentes, en spécifiant un module commun n. Le programme effectue les trois opérations demandées et affiche les résultats. Voici les paramètres d’entrée et les résultats attendus.

1. **Paramètres d’Entrée** :
   * Taille de la première matrice : N1×M1
   * Taille de la deuxième matrice : N2×M2
   * Valeur du modulo : n
2. **Résultats Attendus** :
   * **Affichage des Matrices** : Deux matrices aléatoires de tailles différentes avec des valeurs entre 0 et n−1.
   * **Addition** : Une matrice de dimensions max(N1,N2)×max(M1,M2) où chaque élément est la somme des éléments correspondants des deux matrices, modulo n.
   * **Soustraction** : Une matrice des mêmes dimensions où chaque élément est la différence des éléments correspondants, modulo n.
   * **Multiplication Composante par Composante** : Une matrice où chaque élément est le produit des éléments correspondants, modulo n.

**Tests Limites**

1. **Matrices de Dimensions Différentes** :
   * Nous avons testé des matrices de tailles 3×4 et 3×5 pour vérifier la gestion de tailles différentes.
   * Résultat attendu : Les dimensions de la matrice résultante sont max(3,3)×max(4,5)=3×5.
2. **Modules Différents** :
   * Une exception RuntimeException est lancée si les modules des deux matrices diffèrent.
3. **Valeurs Invalides** :
   * Nous avons vérifié les valeurs d'entrée pour nous assurer que les valeurs dans la matrice sont toujours comprises entre 0 et n−1.
4. **Matrices nul**
   * Nous avons vérifié si nous rentrons une matrice nul le programme s’exécute normalement.
5. **Opération invalide** 
   * Nous vérifions si l’opération n’est pas connue de notre programme que cela crée un RuntimeException

**Factorisation et Extensibilité du Code**

Pour rendre le code extensible, nous avons regroupé les opérations en une seule méthode (operationMatrice). Dans une version future, une approche plus orientée sur les objets, avec des interfaces ou des classes pour chaque type d'opération (addition, soustraction, multiplication) pourrait être envisagée pour supprimer les structures conditionnelles (comme switch) et simplifier l'ajout de nouvelles opérations.

**Exemple de Résultat**

Voici un exemple de sortie attendue pour un test avec N1=3, M1=4, N2=3, M2=5 et modulo = 5.

Une image contenant texte, capture d’écran, Rectangle, conception

Description générée automatiquement

**Conclusion**

L’implémentation de la classe MatriceModulo assure une manipulation correcte des matrices modulo n, et l’encapsulation garantit l’intégrité des données internes. Les opérations sont génériques et pourraient être enrichies ultérieurement sans modifier la structure de base de la classe.

Pour aller plus loin, la factorisation des opérations par des classes spécifiques pourrait améliorer l’extensibilité et la maintenance du code. Le programme a été testé avec divers cas d'usage pour valider la robustesse de l’implémentation.