UFR Sciences et Techniques

Licence d'Informatique
Analyse Numérique

TP n°1 : Bibliothèque matricielle en Java
Patch pédagogique V 1.1

Préambule :
Le but de ce correctif est d'améliorer la description et la standardisation de la gestion des opérations algébriques dans le package `AlgLin`.
Le correctif précise notamment la gestion des erreurs dans les classes `Op*` responsable de cette gestion.
De 1 classe Op dans la version 1.0 du TP, on passe à 3 classes en 1.1 : `Op, OpV` et `IllegalOperationException`.

Classe `Op` et classe `OpV`
Points communs
Ces deux classes *static* implémentent les mêmes méthodes (cf TP 1.0) :

- la somme de deux vecteurs,
- le produit d'un scalaire par un vecteur,
- le produit scalaire de 2 vecteurs,
- le produit d'une matrice par un vecteur colonne,
- la somme de deux matrices,
- la soustraction de deux matrices,
- le produit de deux matrices,
- le produit d'un scalaire par une matrice,
- la division de deux matrices.

Le nom des méthodes est celui de l'opérateur qu'on utiliserait en math pour écrire l'opération. Ainsi, la somme de deux vecteurs sera effectuée par la méthode `Vecteur plus (Vecteur v1, Vecteur v2)`, celle de deux matrices sera `Matrice plus (Matrice m1, Matrice m2)` …
Si le signe opératoire est un '+', la méthode correspondante sera `plus`, si c'est un '.' `point` et un '-' `moins`. La division quant à elle sera notée `loréal`, parce qu'elle le vaut bien.

Différences
Les deux classes divergent sur la réponse à la question : « Que se passe-t-il si quelque chose se passe mal ? ». A ceci `OpV` répond : « Je lève une exception instance de `illegalOperationException` ». `Op` quant à elle répond : « Je ne gère pas cette situation qui doit être traitée en amont ».

Ainsi, si on essaye d'additionner deux vecteurs de dimensions différentes, ou bien de multiplier une matrice et un vecteur dont le nombre de lignes n'est pas le même que celui de colonnes de la matrice, `OpV` lève l'exception susnommée, lors que `Op` ne fonctionnera juste pas.

On pourrait penser qu'alors `Op` n'a que peu d'intérêt. Mais il faut bien voir que la gestion correcte d'un mode dégradé a un coût, celui des tests effectués à chaque appel de méthode. Or, les algorithmes que nous allons utiliser cette année sont très gourmands en calcul : s'épargner des tests très rarement utiles hors période de débuggage est donc une Bonne Chose. De nombreuse opérations élémentaires seront appelées automatiquement dans des longs processus de calcul pour lesquelles les vérifications antérieures devraient suffire à éviter que ces appels en cascade ne lèvent pas d'exception (et ne génèrent pas de multiples vérifications inutiles et couteuses en cascade).

Lorsqu'on utilisera `AlgLin`, on écrira les premières versions du code, celles avec plein de bugs dedans, en utilisant `OpV`, dont les exceptions guideront la mise au point du programme. Une fois que celui-ci s'approchera de la version release, si le temps d'exécution est critique, on remplacera `OpV` par `Op`.

Classe `IllegalOperationException`
Hérite de `Exception`, on écrira essentiellement son constructeur avec un argument en `String`. A l'utilisation Cette `String` sera choisie rationnellement en fonction de la méthode de `OpV` l'appelant.