TP4

Exercice – débogueur

Nous souhaitons utiliser le debogueur pour trouver l'erreur d'un programme et la corriger.

1. Récupérez la classe `Matrix.java`, puis importez-la dans Eclipse. Pour cela, faites un clic-droit sur le répertoire `src` de votre projet et sélectionnez *Import...* puis *File System* dans le dossier *General*.
2. Compilez et exécutez cette classe. L'exécution provoque une erreur. Dans la console Java, un message d'erreur est affiché, indiquant que le programme a accédé à une case d'un tableau avec un index hors limite (ici, 3). Le message d'erreur précise également la ligne de code qui a provoqué l'erreur (ici, la ligne 22), ainsi que la chaîne d'appels de méthodes jusqu'à cette ligne
3. Ajoutez un point d'arrêt à la première ligne de la méthode `main`. Pour cela, faites un clic-droit dans la marge gauche de cette ligne et sélectionnez *Toggle Breakpoint* (ou plus simplement double-cliquez dedans). Le débogueur interrompra alors l'exécution du programme une fois arrivée à cette ligne.
4. Lancez le débogueur en cliquant sur l'icône raccourcie *Debug* (insecte vert) dans la barre d'outils. Lorsque l'exécution du programme atteint un point d'arrêt, Eclipse bascule dans la perspective *Debug* (*i.e.*, le mode débogage).
5. Dans la fenêtre qui s'ouvre pour vous avertir que vous allez changer de perspective, cochez la case *Remember my decision* afin qu'elle ne s'ouvre plus les prochaines fois, et cliquez sur *Yes*.
6. Appuyez sur l'icône *Step Over (F6)* dans la barre d'outils pour passer à la ligne suivante. Appuyez plusieurs fois afin de suivre l'exécution pas à pas du programme et constater l'évolution du contenu des variables (ici, `m1`) dans la vue *Variables*, jusqu'à atteindre la ligne 75 (`Matrice m2 = new Matrice(tab);`).
7. Lorsque le code comporte un appel de méthode (ici, l'appel d'un constructeur), il est possible d'exécuter pas à pas cet appel en appuyant sur l'icône *Step Into (F5)* dans la barre d'outils. Appuyez une fois dessus pour suivre l'exécution du constructeur.
8. Une fois à l'intérieur du constructeur, appuyez plusieurs fois sur l'icône *Step Over (F6)* afin de dérouler le code, et suivez attentivement l'évolution des variables `i` et `j`. Sachant que `m` est un matrice de taille 2×3, `i` doit demeurer entre 0 et 1 et `j` entre 0 et 2. Notez la présence de la référence `this` correspondant à l'objet en construction.
9. Corrigez l'erreur du constructeur.
10. Quittez la perspective *Debug* en cliquant sur la perspective *Java* en haut à droite.
11. Compilez et exécutez de nouveau la classe `Matrix`.
12. Ajoutez un nouveau point d'arrêt à la ligne 22 et relancez le débogueur.
13. Appuyez sur l'icône *Resume* dans la barre d'outils pour poursuivre l'exécution jusqu'au prochain point d'arrêt.
14. Créez un *package* nommé `maths`. Pour cela, faites un clic-droit sur le répertoire `src` de votre projet et sélectionnez *New* puis *Package*.

15. Donnez un nom au *package*, puis cliquez sur *Finish*. Le *package* `maths` apparaît maintenant dans le répertoire `src`.
16. Déplacez la classe `Matrix` dans le *package* `maths`. Le *default package* disparaît.
17. Ajoutez un constructeur par recopie à la classe `Matrix` en utilisant le chainage de constructeurs. S'agit-il d'une recopie profonde ou superficielle ?
18. Ajoutez une méthode `sum()` qui retourne la somme de toutes les valeurs contenues dans la matrice.
19. Ajoutez une méthode `allPositive()` qui retourne `true` si la matrice ne contient que des valeurs strictement positives.
20. Ajoutez une méthode `product(Matrix m)` qui retourne la matrice produit de l'objet receveur et de la matrice passée en paramètre si les dimensions le permettent, sinon retourne `null`.

Exercice – polynôme

Nous souhaitons représenter des polynômes à une variable.

1. Ajoutez un nouveau *package* nommé `polynome` dans le *package* `maths`. Dans votre espace de travail, un répertoire `polynome` a été créé dans le répertoire `maths`.
2. Créez une classe `Polynome` dans le *package* `polynome`. Cette classe stocke les coefficients du polynôme dans un tableau de réels dont le dernier élément est non nul.
3. Écrivez un constructeur qui initialise un polynôme à partir d'un tableau de réels passé en paramètre.
4. Écrivez un accesseur `getDegre()` qui retourne le degré du polynôme (*i.e.*, l'exposant le plus grand parmi ceux des monômes de coefficient non nul).
5. Écrivez un accesseur `get(int i)` qui retourne le coefficient du monôme de puissance `i` (si `i` est négatif ou supérieur au degré du polynôme retourne 0).
6. Écrivez une méthode `toString()`. Par exemple, pour le polynôme $1.7x^5 - 20x + 14$, la méthode `toString` retourne la chaîne de caractères `"1.7*x^5 - 20.0*x + 14.0"`.
7. Écrivez une méthode `equals(Object o)` qui retourne `true` si l'objet receveur et l'objet `o` passé en paramètre possèdent les mêmes coefficients pour chaque monôme, sinon retourne `false`.
8. Écrivez une méthode `evalue(double x)` qui retourne la valeur du polynôme en `x`.
9. Écrivez une méthode `derive()` qui retourne une nouvelle instance de `Polynome` correspondant au polynôme dérivé de l'objet receveur.
10. Cette représentation des polynômes apparaît peu adaptée pour des polynômes de degré important formés de peu de monômes. Par exemple, pour représenter le polynôme $p(x) = 2.2x^{512} - x$ un tableau de 513 réels sera créé alors que le polynôme ne comporte que 2 monômes. Nous proposons de changer de représentation en utilisant un tableau de monômes. Ajoutez une classe `Monome` dans le *package* `polynome`, sachant qu'un monôme est représenté par un exposant et un coefficient.
11. Ajoutez un constructeur et des accesseurs à cette classe.
12. Modifiez la classe `Polynome` afin d'utiliser un tableau de `Monome` à la place d'un tableau de `double`. L'ordre des monômes dans le tableau est quelconque (*i.e.*, pas forcément triés par ordre croissant de puissance).
13. Donnez les nouvelles implémentations des méthodes qu'il est nécessaire de réimplémenter.