TD Concepts objet et Java

2016-17

Exercice 3

On souhaite développer des classes Java permettant de faire certaines manipulations sur des nombres complexes. Pensez, au fur et à mesure de vos développements à tester de façon unitaire chaque nouvelle unité de code implémentée.

1. Définissez une classe `Complexe` comprenant deux champs de type `double`, représentant respectivement la partie réelle et la partie imaginaire d'un nombre complexe, puis munissez-la d'un constructeur prenant une valeur pour la partie réelle et la partie imaginaire du nouveau nombre complexe.
2. Ajoutez à la classe `Complexe` le code nécessaire pour permettre un affichage approprié pour des objets de cette classe.
3. On ne souhaite pas que les objets de la classe `Complexe` puissent être modifiés une fois créés (on les qualifiera donc d'*immuables* (*immutable*)). Comment peut-on garantir cela ? Implémentez la partie pertinente des méthodes d'accès.
4. Ajoutez un *constructeur de copie* à votre classe prenant en unique paramètre une instance de la classe `Complexe`. Quelle peut être dans ce contexte l'utilité d'un tel constructeur ?
5. Ajoutez le code nécessaire à votre classe pour tester l'égalité d'état de deux objets de la classe `Complexe` selon la méthode préconisée pour le langage Java.
6. Ajoutez des méthodes d'instance pour le calcul du module et de l'argument d'un complexe. On rappelle :
   — $module = \sqrt{re^2 + im^2}$
   — $argument = acos(\frac{re}{module})$
7. Définissez des méthodes pour l'addition et la multiplication de deux nombres complexes. Ces méthodes prendront un paramètre implicite et un paramètre explicite, et retourneront une nouvelle instance correspondant au résultat de l'opération. On rappelle que :
   — $(re_1 + im_1 i) + (re_2 + im_2 i) = (re_1 + re_2 + (im_1 + im2)i)$
   — $(re_1 + im_1 i) * (re_2 + im_2 i) = (re_1 * re_2 - im_1 * im_2 + (re_1 * im_2 + im_1 * re_2)i)$
8. On souhaite à présent bénéficier de la classe `Complexe` et définir une classe permettant de représenter un nombre complexe telle que l'historique des opérations dans lesquelles ce complexe aura servi d'opérande est conservé. Définissez une nouvelle classe `ComplexeMemoire`, dans un autre package que `Complexe`, permettant de réaliser cela. Pour l'historique des opérations, on veut garder la trace des opérations subies (addition ou multiplication), de la valeur des autres opérandes et des résultats obtenus. Proposez une implémentation appropriée, et ajoutez un constructeur prenant une partie réelle et une partie imaginaire en paramètres.
9. Ajoutez un constructeur par copie à la classe `ComplexeMemoire`.

10. Serait-il possible d'ajouter simplement un constructeur sans paramètre à la classe `ComplexeMemoire` ? Si l'on se contente de laisser vide le bloc de ce constructeur, quel problème va-t-on rencontrer ?
11. Ajoutez à la classe `ComplexeMemoire` les méthodes nécessaires pour pouvoir ajouter des messages à la mémoire des opérations d'un objet de la classe et consulter cette mémoire.
12. Proposez à présent une redéfinition adaptée (utilisez l'annotation `@Override`) dans la classe `ComplexeMemoire` des méthodes d'addition et de multiplication de complexes définies dans la classe `Complexe`. Est-il possible et utile d'adapter le type de retour (*covariance*) ?
13. On souhaite à présent mettre en place une mémoire *collective* où apparaît une seule fois chaque opération effectuée sur des instances de `ComplexeMemoire`. Adaptez votre classe afin de permettre cela.
14. Sans modifier le code développé jusqu'à présent, que se passera-t-il si l'on invoque la méthode `equals` sur deux instances de la classe `ComplexeMemoire` ? Sur une instance de la classe `Complexe` et une instance de la classe `ComplexeMemoire` ?
15. On souhaite à présent pouvoir sauvegarder sur disque l'état d'un objet de la classe `ComplexeMemoire` pour pouvoir le restaurer plus tard. Pour cela, vous allez avoir recours à la technique de *sérialisation* (*serialization*) (voir par exemple : `https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serial-arch.html`). Vous pourrez à cette fin opérer des modifications à la classe mère `Complexe`, mais on ne souhaite pas rendre cette dernière elle-même directement *sérialisable*. Testez votre solution en effectuant le chargement de l'état d'un objet de type `ComplexeMemoire` sauvegardé sur disque.
16. La documentation suggère l'utilisation explicite d'un champ `serialVersionUID` (`static final long`) pour renseigner le processus de sérialisation sur la version d'une classe. Que se passe-t-il si l'on chercher à *désérialiser* un objet avec une définition de classe incompatible avec celle utilisée dans un fichier de *sérialisation* particulier ?
17. Comment faire si l'on décidait que la mémoire d'un objet de type `ComplexeMemoire` ne devait pas faire partie des informations sauvegardées avec l'état d'un objet ? Implémentez et testez votre solution.
18. Qu'en est-il des champs `static` de la classe en lien avec les objets sérialisés ?