Feuille d'exercices N°1

Exercice 1 *Nombres complexes*

Dans ce TD, il s'agit d'implémenter la classe `Complexe` qui représente un nombre complexe.

1 Structure et Interface de la classe Complexe

La classe `Complexe` considérée dans ce TD correspond aux spécifications suivantes.

1.1 Variables d'instances

On représente un nombre complexes par deux variables d'instance de type `double` : `x` et `y` pour représenter respectivement la partie réelle et imaginaire du nombre complexe.

Remarque 1 *Les **variables d'instances** sont **propre à chaque objet** de la classe : la modification d'une variable d'instance d'un objet ne modifie pas la valeur de la variable correspondante d'un autre objet.*

Il vous est demandé de mettre le bon mot clé de visibilité, de sorte que ces champs ne puissent être accessibles que de la classe `Complexe` (se reporter aux mots clés de visibilité dans les notes de cours).

Remarque 2 *En théorie il est recommandé d'avoir le moins possible de variables de type* `public` *afin de pouvoir modifier par la suite la programmation d'une classe sans modifier l'interface avec les autres classes (voir accesseurs (getter) et modifieur (setter) dans les notes du cours).*

1.2 Constructeur

Le ou les constructeurs permettent d'initialiser les variables d'instances et en particulier de "construire" les variables d'instance de type objet.

Pour construire un nombre complexe nous voulons donner trois possibilités différentes à l'utilisateur :
- `public Complexe()` : construction du nombre complexe nul avec un constructeur sans paramètre
- `public Complexe(double x)` : construction d'un nombre réel pure en utilisant un constructeur avec un paramètre
- `public Complexe(double x,double y)` : construction d'un nombre complexe avec un constructeurs à deux paramètres.

Remarque 3 *L'utilisation du mot clé* `this` *permettra de spécifier au compilateur que la variable utilisée est la variable d'instance de l'objet appelant. Cela permet d'avoir les même noms pour les paramètres du constructeur que pour les variables d'instance, ce qui en général est plus clair.*

Remarque 4 *Si vous ne programmez pas de constructeurs dans la classe* `Complexe` *(et seulement dans ce cas), Java créera automatiquement un constructeur par défaut sans paramètre. Le constructeur par défaut se contente d'initialiser les variables d'instance à* `zero` *(*`null` *pour les objets le cas échéant).*

1.3 Méthodes d'instance

Pour manipulez les nombres complexes il faut programmer des méthodes d'instances. Ces méthodes permettent soit d'agir sur le nombre complexe appelant soit de renvoyer le résultat d'un calcul effectué à partir du nombre complexe appelant et des paramètres.
- `public Complexe plus(Complexe z)` : calcule la somme de deux complexes,
- `public void conjugue()` permet de conjuguer un complexe
- `public double norme2()`permet de calculer le carré de la norme d'un complexe

1.4 Variable de classe

Parmi les nombres complexes il en est un qui est plus particulièrement utilisé : l'imaginaire pure `i`. Afin de pouvoir l'utiliser sans avoir à le recréer nous créons une variables de classes contenant `i` :

```
public static Complexe i=new Complexe(0,1);
```

Cette variable est déclarée de préférence au début de la classe (à côté des variables d'instance).
- Le mot clef `static` indique que c'est une variable de classe : il en existe une seule instance commune à tous les objets de type complexe.
- Pour accéder à la variable de classe `i` il faut écrire `Complexe.i`. En général pour accéder à une variable de classe il faut écrire : `NomDeLaClasse.nomDeLaVariable`.

1.5 Méthode de classe

Dans le cas des nombres complexes nous n'avons pas réellement besoin de méthode générique. Néammoins il peut être utile de disposer d'une méthode renvoyant un nombre complexe à partir des coordonnées polaires. Etant donné qu'il est impossible d'écrire un nouveau constructeur prenant en paramètre deux nombres réels nous allons utiliser une méthode de classe :

```
public static Complexe polaire(double r,double theta) {
        return new Complexe(r*Math.cos(theta),r*Math.sin(theta)) ;
}
```

Pour utiliser une méthode de classe il faut écrire `NomDeLaClasse.nomDeLaMethode(...)`.

Remarque 5 *Les deux coordonnées polaires r et θ peuvent être converties en coordonnées cartésiennes x et y en utilisant les fonctions trigonométriques sinus et cosinus :*
$x = r\cos\theta$
$y = r\sin\theta$

2 Utilisation de la classe Complexe : La méthode main

Testez la classe `Complexe` à l'intérieur de la méthode `main` afin de réaliser les traitements suivants :

1. déclaration et construction de trois nombres complexes `a=0`, `b=1.0`, `c=1.0+i` ;
2. affichage de `a`, `b` et `c` ;
3. modification des variables d'instances de `a` (`x=-1` et `y=-2`)
4. conjugaison de `c` et nouvel affichage de `c` ;
5. calcul de `c+i` et affichage du résultat
6. stockage de `c+i` dans `a` et affichage de `a`
7. création et affichage du nombre complexe $2 * e^{(i\pi/2)}$
8. affichage de la norme de `d` au carré

Rappel de formules :
Un nombre complexe z se présente en général en coordonnées cartésiennes, comme une somme $a + b$i, où a et b sont des nombres réels quelconques et i (l'unité imaginaire) est un nombre particulier tel que $\mathrm{i}^2 = -1$.
– Le réel a est appelé partie réelle de z et le réel b est sa partie imaginaire.
– Pour **additionner deux nombres complexes**, on additionne séparément leurs parties réelles et imaginaires : $(a + b\mathrm{i}) + (c + d\mathrm{i}) = (a + c) + (b + d)\mathrm{i}$;
– **Conjugué d'un nombre complexe**. Le nombre complexe conjugué de $z = a + b$i est le nombre complexe $\overline{z} = a - b\mathrm{i}$;
– Le **carré de la norme** d'un nombre complexe z est $norme2 = a^2 + b^2$ avec a et b les parties réelle et imaginaire du complexe z, respectivement.
– La **forme trigonométrique** d'un nombre complexe est :
$z = r(cos(\theta) + isin(\theta))$ avec r réel tel que $r > 0$.
La formule d'Euler $e^{i\theta} = cos(\theta) + isin(\theta)$ permet de compacter cette écriture sous une forme exponentielle $z = re^{i\theta}$.

Exercice 2 *Compte Bancaire*

Dans cet exercice nous utilisons la classe `CompteBancaire` dont l'interface est la suivante.

3 Structure et Interface de la classe CompteBancaire

3.1 Variables d'instance

Les comptes bancaires `CompteBancaire` sont représentés le plus simplement possible, i.e., par deux variables d'instance de noms respectifs `solde` et `id` :
– `solde` : un nombre en double précision conservant le solde du compte ;
– `id` : une chaîne de caractères mémorisant l'identité du propriétaire du compte.

3.2 Constructeurs

On dotera cette classe des 2 constructeurs
– `CompteBancaire(double soldeInitial, String idInitial)` constructeur avec valeurs initiales explicites pour le `solde` et le nom `id` du propriétaire du compte.
– `CompteBancaire()` constructeur sans arguments, donnant la valeur `0` au `solde`, et la valeur `"anonyme"` au nom du propriétaire ;

3.3 Accesseurs et modifieurs

- `String getId()` accesseur en lecture retournant une chaîne de caractères désignant le propriétaire du compte ;
- `void setId(String id)` accesseur en écriture permettant de changer le propriétaire du compte, du moins la chaîne de caractères le représentant ;
- `double getSolde()` accesseur en lecture retournant le montant du solde ;
- `void setSolde(double solde)` accesseur en écriture permettant de changer le montant du solde.

3.4 Méthodes d'instance

- `void retirer(double montant)` opération débitant le compte d'un montant strictement positif `montant` si le solde du compte est au moins égal à `montant` (ne fait rien sinon) ;
- `void deposer(double montant)` opération créditant le compte d'un montant strictement positif `montant` (ne fait rien si `montant` n'est pas strictement positif) ;
- `void transferer(double montant, CompteBancaire compteDepot)` opération transférant le montant strictement positif `montant` du compte courant au compte `compteDepot` si le solde du compte courant est au moins égal à `montant` (ne fait rien sinon) ;
- `void afficher()` affiche à l'écran la chaîne de caractères caractérisant l'état du `CompteBancaire` ;

4 Utilisation de la classe CompteBancaire

1. Déclarer une référence `compte` sur un `CompteBancaire` ;
2. créer/construire le compte bancaire correspondant ;
3. affecter `Mohamed Kamel` comme propriétaire du compte, déposer `100` unités, puis encore `50` unités, en retirer `20` pour acheter des livres, calculer le solde ;
4. en supposant qu'il existe un autre compte référencé par la variable `compteEpargne`, transférer `100` unités du compte `compte` au compte `compteEpargne`.

Exercice 3 *Tableau d'entiers*

Un tableau d'entiers est caractérisé par sa taille effective et le tableau de ses composants (au plus 100 éléments). La classe possède les méthodes suivantes :

1. une méthode `initialiser` qui initialise les éléments du tableau ;
2. une méthode `afficher` qui affiche les éléments du tableau d'entiers ;
3. une méthode `ajouter` qui ajoute un élément passé en paramètre au tableau. La méthode ne fera rien si le tableau est déjà plein ; en ce cas elle renverra le booléen `false` ;
4. une méthode `supprimer` qui supprime un élément du tableau (le premier élément du tableau égal à la valeur du paramètre de la méthode). Il faudra "tasser" les éléments non vides si l'élément supprimé est au milieu des éléments non vides. La méthode renvoie l'indice du tableau de l'élément qui a été supprimé. Elle renvoie `-1` si l'élément n'a pas été trouvé ;
5. une méthode `testTri` qui vérifie si le tableau est trié en ordre croissant ou non ;
6. une méthode `tri` qui trie les éléments du tableau dans l'ordre croissant ;

7. une méthode `maxTableau` qui renvoie le plus grand entier du tableau d'entiers ;
8. une méthode `minTableau` qui renvoie le plus petit entier du tableau d'entiers ;
9. une méthode `rechercheSeq` qui effectue une recherche séquentielle d'une valeur donnée en paramètre ;
10. une méthode `rechercheDicho` qui effectue une recherche dichotomique d'une valeur donnée en paramètre ;

Définir la classe `Tableau` qui répond à la description donnée et écrire une classe `ProgPrincipale` qui permet de tester ces méthodes en affichant un menu où l'on peut choisir une des opérations citées ci-dessus appliquées à un tableau d'entiers donné.

Exercice 4 *Réalisation d'une pile d'entiers au moyen d'un tableau*

Définir une classe `PileTableau` décrite par un tableau d'entiers (utiliser la classe `Tableau` définie dans l'exercice 3).

Les attributs associés à cette classe seront :
– une taille maximale pour le tableau stockant les éléments de la pile
– un tableau d'entiers destiné à stocker les éléments de la pile
– un entier sommet donnant l'indice du tableau correspondant au sommet de la pile (initialisé à -1 par exemple)

Vous associerez à votre classe un constructeur prenant en paramètre la taille maximale associée à la pile. Puis vous la doterez de l'ensemble des méthodes suivantes :
– `public void affiche()` : afficher des éléments de la pile,
– `public void empiler (int elt)` : rajouter un élément dans la pile,
– `public void depiler()` : supprimer un élément de la pile,
– `public boolean pileVide()` : retourne `true` si la pile est vides `false` sinon,
– `public boolean pilePleine()` : retourne `true` si la pile est pleine `false` sinon,
– `public int sommet()` : retourne la valeur qui se trouve au sommet.

Écrire un programme qui étant donné une pile d'entiers détermine la valeur maximale et la supprime de la pile (supprimer aussi toutes las valeurs égales à la valeur maximale).

Bon travail !