

ALGORITHMIQUE III.2

TYPES, CLASSES, INSTANCES

À propos des exercices individuels Les exercices notés (*) sont obligatoires et individuels. Ils sont à faire avant le prochain *cours*. Ils seront notés. Une réponse fautive, sans commentaires et rendue par plusieurs personnes sera considérée comme de la copie et vaudra 0.

En d'autres termes, *testez vos programmes* !

À propos des exercices collaboratifs Les exercices notés © sont aussi obligatoires et partagés entre toute la classe. Ils sont aussi à faire avant le prochain cours. Ils seront aussi notés mais la note sera appliquée à toute la classe.

Interfaces (Patron de conception *Interface*)

Pour ce qui suit, nous allons employer l'interface `ComplexNumber`, définie de la manière suivante :

```
//Fichier fr/univ_orleans/mi2007/java/td2/complex/ComplexNumber.java
package fr.univ_orleans.mi2007.java.td2.complex;

public interface ComplexNumber
{
    /**
     * Renvoie la partie réelle de ce nombre
     */
    public float getReal();
    /**
     * Renvoie la partie imaginaire de ce nombre
     */
    public float getImaginary();
    /**
     * Renvoie le module de ce nombre
     *
     * @return Un nombre positif ou nul.
     */
    public float getModule();
    /**
     * Renvoie l'argument de ce nombre.
     *
     * Si ce nombre est égal à 0, le résultat de cette méthode n'est pas
     * spécifié mais ne peut pas être NaN. Par convention et pour diminuer les
     * erreurs d'arrondi, le résultat est toujours dans  $[0, 2 * \text{Math.PI}[$ 
     *
     * @return Un nombre compris dans  $[0, 2 * \text{Math.PI}[$  .
     */
    public float getArgument();
    /**
     * Change la valeur de ce nombre complexe
     *
     * @param real La valeur à donner à la partie réelle.
     * @param imaginary La valeur à donner à la partie imaginaire.
     */
    public void setCartesian(float real, float imaginary);
    /**
```

```

    * Change la valeur de ce nombre complexe
    *
    * Le résultat de cette fonction n'est pas défini si le module
    * est négatif.
    *
    * @param module Un nombre positif ou nul, représentant la valeur à
    * donner au module
    * @param argument Un nombre quelconque.
    */
    public void setPolar(float module, float argument);
}

```

Exercice 1. Implantez l'interface `ComplexNumber` ci-dessus sous la forme d'une classe `CartesianComplexNumberImpl`, qui utilise la représentation cartésienne des nombres complexes.

Exercice 2. Implantez cette interface sous la forme d'une classe `PolarComplexNumberImpl`, qui utilise la représentation polaire des nombres complexes.

Considérons maintenant l'interface `ComplexUtilities` suivante :

```

//Fichier fr/univ_orleans/mi2007/java/td2/complex/ComplexNumber.java
package fr.univ_orleans.mi2007.java.td2.complex;

public interface ComplexUtilities
{
    /**
     * Calcule la distance entre deux nombres complexes.
     *
     * @param a Un nombre complexe (ne peut pas être null)
     * @param b Un nombre complexe (ne peut pas être null)
     *
     * @return Un nombre positif ou nul.
     */
    public static float getDistance(Complex a, Complex b)
}

```

Exercice 3. Implantez cette interface dans une classe `ComplexUtilitiesImpl`. Testez cette fonction avec les deux implantations de `ComplexNumber`.

Exercice 4. L'extrait suivant contient des erreurs, corrigez-les.

```

//Fichier td2/acorriger.java
package fr.univ_orleans.mi2007.java.td2.comparaisons;

/**
 * Une interface implantée par tous les objets qu'on peut comparer
 * deux à deux. Cette interface sera utilisée notamment par les
 * algorithmes de tri.
 */
public interface ObjetComparable
{
    /**
     * Compare cet objet à un autre
     *
     * @param o Un objet non-null auquel comparer cet objet.
     *
     * @return 1 si cet objet est supérieur ou égal à o,

```

```

    * 0 si cet objet est égal à o, -1 sinon.
    */
public static final int compare(ObjectComparable o)
{
    if(this < o)
        return -1;
    else if(this >= o)
        return 1;
    else if(this = o)
        return 0;
}
}

```

Et chez vous...

Chaînes de caractères

Exercice 5. * Pour ce qui suit, vous aurez besoin de consulter la documentation de Java, disponible sur le site de JavaSoft.

Que fait la méthode `String.charAt` ? Que fait la méthode `String.length` ? À l'aide de ces méthodes, implantez la fonction suivante

```

/**
 * Renvoie le dernier caractère de s
 */
public static char getLastChar(String s)

```

N'oubliez pas de tester !

Tableaux

```

//Fichier fr/univ_orleans/mi2007/java/td2/arrays/AlmostAnArray.java
package fr.univ_orleans.mi2007.java.td2.arrays;

```

```

/**
 * Une interface pour une classe qui dispose des primitives principales
 * du tableau.
 */
public interface AlmostAnArray
{
    /**
     * Renvoie la longueur de ce tableau.
     *
     * @return Un entier positif ou nul.
     */
    public int getLength();
    /**
     * Renvoie un élément du tableau.
     *
     * Renvoie null si l'indice est strictement négatif ou supérieur ou égal
     * à la longueur du tableau.
     *
     * @param index Un entier positif ou nul et strictement inférieur à la
     * longueur du tableau, et qui détermine la position à consulter dans
     * le tableau.
     * @return Le premier élément du tableau si l'index vaut 0, le deuxième
     * si l'index vaut 1... Ou null si l'index n'est pas dans [0,longueur[

```

```

    */
public String getElementAt(int index);
/**
 * Modifie un élément du tableau.
 *
 * @param index Un entier positif ou nul et strictement inférieur à la
 * longueur du tableau et qui détermine la position à modifier dans le
 * tableau.
 * @param element Le nouvel élément à mettre dans le tableau à l'emplacement
 * spécifié par l'index.
 *
 * @return L'élément qui était précédemment à cet emplacement dans le
 * tableau, ou null s'il n'y avait aucun élément à cet emplacement dans
 * le tableau.
 */
public String setElementAt(int index, String element);
}

```

Note Pour rendre cette interface utilisable, il faudrait en fait déclencher une erreur lorsque l'utilisateur essaye de lire ou d'écrire une valeur qui n'existe pas dans le tableau. Nous verrons plus tard comment faire ceci à l'aide du mécanisme des *exceptions*.

Exercice 6. * Implantez l'interface `AlmostAnArray` précédente sous la forme d'une classe `AlmostAnArrayImpl`, qui utilisera en interne un tableau de chaînes de caractères.

N'oubliez pas de commenter vos méthodes.

Quelles autres implantations pourrait-on imaginer ?

Exercice 7. © Mettez les réponses aux exercices 1 à 4 sur le Wiki, y compris ceux qui n'ont pas été corrigés en classe.

Exercice 8. © Mettez le contenu du cours sur le Wiki, y compris la fin du corrigé aux exercices de la semaine dernière. Vous êtes encouragés à redécouper et à reformuler plus clairement.