TP1 : Premiers pas en Java

Le but de ce TP est de vous familiariser avec la syntaxe de base de Java à travers des exemples simples.

1 Recommandations

Pendant tout ces travaux, et les suivants, une attention particulière sera portée à ce que toute classe apparaisse dans un *package* nommé, et toute classe soit déclarée soit abstraite (`abstract`) soit verrouillée (`final`).

N'hésitez pas à fouiller la bibliothèque standard dont la documentation est accessible sur http://docs.oracle.com/javase/7/docs/api/. Voyez comme cette documentation est pratique ; vous pouvez faire aussi bien : l'outil `javadoc`[1] extrait la documentation de vos commentaires. Essayez-le et commentez chaque classe, chaque méthode, chaque attribut.

N'hésitez pas à abuser du mot clef `final`.

Chaque classe demandée sera codée dans un fichier séparé, dont le nom sera de la forme `Nom.java`. Chaque fichier d'un *package* devra être placé dans un dossier au nom dudit *package*. Les sources seront placées dans un dossier `src`, et l'ensemble des fichier `.class` dans un dossier `bin`. La compilation d'un fichier source du *package* `algebre` se fera donc depuis le dossier parent de `src` et `bin` par la commande :

```
javac -d bin -cp bin src/algebre/Nom.java
```

Pour compiler l'ensemble du *package* `algebre`, vous pourrez utiliser :

```
javac -d bin -cp bin src/algebre/*.java
```

L'exécution de la fonction `main` de la classe `Nom` du *package* `pack` se fera depuis le dossier principal comme suit :

```
java -cp bin/ pack.Nom
```

2 Nombres complexes

Créez une classe *abstraite* `Complex` dont les instances représentent les nombres complexes. Cette classe aura les quatre méthodes suivantes, dont l'implémentation peut être remise à plus tard :

— `re()` qui renvoie la partie réelle de ce complexe ;
— `im()` qui renvoie la partie imaginaire de ce complexe ;
— `mod()` qui renvoie le module de ce complexe ;
— `arg()` qui renvoie un argument de ce complexe (à préciser).

1. Plus d'info sur : http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/javadoc.html

Définissez deux classes `CartesianComplex` et `PolarComplex` qui spécialisent la classe `Complex` en choisissant une représentation des nombres complexes (cartésienne ou polaire, respectivement). Elles disposeront notamment d'un constructeur à deux arguments.

Ajoutez les méthodes suivantes à la classe `Complex` :

— `conjugate()` qui renvoie le conjugué du complexe courant ;
— `opposite()` qui renvoie l'opposé du complexe courant ;
— `add(Complex c)` qui renvoie la somme du complexe courant et de `c` ;
— `mul(Complex c)` qui renvoie le produit du complexe courant et de `c` ;
— `div(Complex c)` qui renvoie le quotient de la division du complexe courant par le complexe `c` (cette méthode doit lever une exception si le dénominateur de la division est nul) ;
— `div(double d)` qui renvoie le quotient de la division du complexe courant par le réel d ;
— `toString()` ;
— `hashCode()` et `equals(Complex c)`

Ajoutez la constante *i* à la classe `Complex` :

```
public static final Complex I = ...
```

Les instances de la classe `Complex` sont-elles mutables ? Si ce n'est pas le cas, modifiez vos classes dans ce sens. Ajoutez une méthode `conjugateMe()` qui modifie le complexe courant (le change en son conjugué). Est-ce une bonne idée ? Que se passe-t-il si l'on exécute l'instruction suivante ?

```
Complex.I.conjugateMe();
```

3 Booléens

Proposez une classe (abstraite) `Bool` et deux objets, `TRUE` et `FALSE`, pour représenter les valeurs de vérité.

Programmez une fonction (statique) `ofBoolean` qui permet de réifier un booléen Java. Programmez quelques méthodes de manipulation de booléens : `not`, `and`, `or`…

4 Matrices de complexes

Programmez une classe `ComplexMatrix` qui permette de créer des matrices de complexes de taille quelconque. Vous implémenterez la multiplication et l'addition de matrices.

Vous gèrerez les contraintes de dimension des matrices en créant un nouveau type d'exception `MatrixDimensionMismatchException`.

De plus, vous implémenterez une méthode (nommée `withCoeff`) permettant de modifier un coefficient de la matrice courante.

Enfin, une méthode `isSymmetric()` vous permettra de déterminer si une matrice est symétrique.