TD 3
Objectif : classes génériques, interfaces et itérateurs

Nous proposons de coder un itérateur pour effectuer différents parcours d'une matrice à deux dimensions, comprenant des valeurs de type entier.

Exemple de matrice de 3 lignes et 4 colonnes :
1 2 3 4
5 6 7 8
9 10 11 12

Une interface :

L'interface `java.util.Iterator<E>` comprend les méthodes suivantes :

```
boolean hasNext();
// retourne true s'il reste des éléments à parcourir dans la collection
E next();
// retourne le prochain élément du parcours
void remove();
// supprime de la collection le dernier élément parcouru
```

La méthode `remove()` est une méthode optionnelle, nous l'écartons du sujet.
Nous demanderons donc de coder un itérateur qui puisse gérer des parcours sur un tableau d'entiers à deux dimensions, à l'aide des deux méthodes suivantes :

```
  boolean hasNext() ;
// retourne true s'il reste des éléments à parcourir dans la collection
  E next() ;
// retourne le prochain élément du parcours.
```

Pour simplifier le sujet nous ne demandons pas de gérer des exceptions.

Trois parcours possibles :
Une matrice peut être parcourue de 3 façons différentes, comme le montre l'exemple ci-dessous.
Matrice :
1 2 3 4
5 6 7 8
9 10 11 12

L'affichage des valeurs avec un parcours en ligne :
1 2 3 4 5 6 7 8 9 10 11 12
L'affichage des valeurs avec un parcours en colonne :
1 5 9 2 6 10 3 7 11 4 8 12
L'affichage des valeurs avec un parcours en zig-zag :
1 2 3 4 8 7 6 5 9 10 11 12

Afin de pouvoir réaliser les trois parcours on prévoit un itérateur pour chacun des parcours comme proposé dans l'interface suivante. La méthode `getIterator` permettra de retourner l'itérateur dédié au type de parcours souhaité, selon la valeur du paramètre.

```
public interface MatrixIterable <E> {
 /**
       * @param n : n = 1 si itérateur pour parcours lignes par lignes
       * n= 2 si itérateur pour parcours colonnes par colonnes
       * n= 3 si itérateur pour parcours en zig zag
        * Retourne un itérateur pour parcourir l'instance courante
        * selon le parcours donné par le paramètre n
       */
public Iterator<E> getIterator(int n) ;
}
```

1) Pour commencer, et **indépendamment de cette interface**, coder une classe `IntegerMatrix` qui modélise une matrice à deux dimensions. Les constructeurs permettent de générer des matrices de taille aléatoire, d'entiers aléatoires positifs ou nuls. Fournir les *getters* utiles et le *toString()*. Coder un programme principal test.

2) Donner sous la forme d'un diagramme, les classe(s) et interface(s) utiles pour générer les itérateurs qui seront retournés par la méthode :
`Iterator<E> getIterator(int n).`
N.B. On peut modéliser par des classes externes ou des classes internes à la matrice. Explorez les deux modélisations.

3) Implanter ces classes et faire implanter l'interface `MatrixIterable<E>` à la matrice.

4) Compléter le programme principal pour tester les trois parcours proposés. La fonction *main()* affichera les valeurs d'une matrice d'entiers successivement par lignes, puis par colonnes, puis en zig zag.
5) Faire implanter à la classe `IntegerMatrix`, en plus de l'interface (personnelle) `MatrixIterable<Integer>`, l'interface java standard `Iterable<Integer>` : choisir par exemple l'itérateur ligne comme itérateur par défaut.
6) Expérimentez dans la fonction *main()* de parcourir la matrice au moyen d'une « boucle for each »