TP2 : Mots croisés avec grilles génériques et itérateurs

1 Grilles génériques

1.1 Classe `Grille` générique

Créez un nouveau paquetage pour votre TP2 (par exemple `binome1.binome2.tp2`), copiez-y les classes `Grille` et `MotsCroises` réalisées pour le TP1. Modifiez votre classe `Grille` afin de la rendre générique, comme dans cet exemple d'utilisation :

```
Grille<String> maGrille = new Grille<String>(3,5) ;
for (int l=1; l<=maGrille.getHauteur(); l++)
{
    String texteLigne = Integer.toString(l);
    for (int c=1; c<=maGrille.getLargeur(); c++)
    {
        maGrille.setCellule
            (l, c, texteLigne + ',' + Integer.toString(c));
    }
}
System.out.println(maGrille) ;
```

Le résultat affiché doit être identique à celui demandé pour l'exercice 1 du TP1.
Modifiez ensuite votre classe `MotsCroises` en conséquence. Vérifiez la non-régression à l'aide du test JUnit `MotsCroisesTest`.

1.2 Classe `MotsCroisesAvecHeritage`

Créez puis testez une copie de la classe `x.y.tp2.MotsCroises` appelée `MotsCroisesAvecHeritage`, en faisant en sorte que celle-ci hérite de la classe `x.y.tp2.Grille`. Il vous faudra déclarer à cet effet une nouvelle classe qui regroupe en un seul objet les quatre informations d'une case de mots croisés : solution, proposition, définition horizontale et définition verticale (on pourrait envisager aussi un cinquième champ booléen `caseNoire`). Vérifiez la non-régression à l'aide de `MotsCroisesTest`.

2 Parcours des solutions/propositions/définitions par des itérateurs

2.1 Itérateur de "mots" d'un vecteur

Créez une classe non générique `IterateurMots` implémentant l'interface `Iterator<String>`. En plus des méthodes à implémenter, vous déclarerez :

— une variable d'instance de type `Object[]` ;

— une variable entière servant de curseur pour ce tableau ;
— un constructeur permettant d'initialiser ces deux variables : la première à l'aide d'un unique paramètre, et la deuxième par l'indice du premier élément non `null` du tableau, ou l'indice du dernier élément + 1 s'ils sont tous `null`.

Voici comment implémenter les méthodes de l'interface :

— `remove()` se contente de faire `throw new UnsupportedOperationException()` suivant le principe de la méthode optionnelle, car votre itérateur ne peut enlever des éléments ;
— `hasNext()` retourne `true` si et seulement si le curseur pointe sur un élément existant (inférieure ou égale à l'indice du dernier élément donc) ;
— si l'on a `hasNext()`, `next()` retourne la chaîne de caractères obtenue par concaténation des valeurs successives des éléments du tableau, de l'élément pointé par la valeur courante du curseur, jusqu'au prochain élément `null`, non inclus. Avant de retourner cette valeur, il faudra déplacer le curseur sur le premier élément non `null` suivant, ou à défaut sur l'indice du dernier élément + 1.

Ainsi la séquence suivante :

```
Character[] tab = new Character[8];
tab[0] = null;
tab[1] = 'A';
tab[2] = 'B';
tab[3] = null;
tab[4] = null;
tab[5] = 'C';
tab[6] = 'D';
tab[7] = null;
IterateurMots iter = new IterateurMots(tab) ;
while(iter.hasNext())
{
    System.out.print(iter.next() + ", ") ;
}
```

doit afficher ceci :

```
AB, CD,
```

2.2 Itérateur de ligne/colonne de grille

Écrivez la méthode `public IterateurMots iterateurMots(boolean horizontal, int num)` de la classe `GrilleGen` retournant un itérateur sur la ligne ou la colonne spécifiée par ses paramètres.

2.3 Fonctions de parcours des lignes/colonnes d'un `MotsCroisesAvecGen`

Écrivez la méthode `public String solutions()` de la classe `MotsCroisesAvecGen` retournant la solution complète de la grille, à raison d'une ligne de texte par ligne de la grille, et d'un mot par séquence de lettres consécutives, le tout précédé du numéro de ligne.