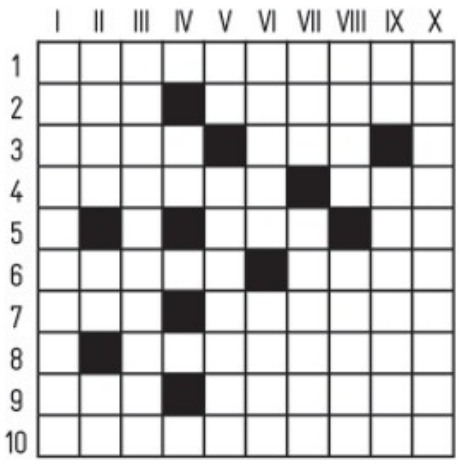
Programmation orientée objet
ING 1
une grille de mots croisés 2ème partie

Objectif

L'objectif de ce TD/TP est de gérer une grille de mots croisés en utilisant la POO. Vous connaissez tous le principe des mots croisés (wikipedia si vous ne connaissez pas). Nous allons au cours de plusieurs TD/TP nous familiariser avec l'écriture de mots croisés en Java

MOTS CROISÉS PAR JEAN-FRANÇOIS DEMAY

GRILLE N° 48



HORIZONTAL :
1. Dans quelques semaines, ce sera une bonne occasion de se rendre dans une boîte interdite aux mineurs. **2.** Boursouflé chez les grosses têtes. Phosphore. **3.** Défunts depuis peu. Petite montagne. **4.** Devient huile quand elle est grosse. Méfiance quand elle dort ! **5.** Espion burlesque. Diminutif de lady. **6.** Paresseux. Un jeune qui porte des sabots. **7.** Grabuge. Safari pour John Wayne. **8.** Conçois des chimères. **9.** Roulée. Alcoolisant le rouge. **10.** Comme une situation à France Télécom.

VERTICAL :
I. Ainsi sont les tombes à la Toussaint. **II.** Suicidaire grec. Conjonction. Vieille note. **III.** Paltoquet. **IV** Connu. **V.** Vieille langue. Coopératives agricoles en Israël. **VI.** Écrits sur un cours. Animés. **VII.** Homme de cour chez les Ottomans. Rendit tout doux. **VIII.** Prénom révolutionnaire. Sucrerie populaire. **IX.** Coordonne. Vouent un culte. **X.** Un politicien qui s'habille en noir.

Solution de la grille n° 47 :

1. Bling-bling
2. Oulan-Bator
3. Aïeul. Rêva
4. Trot. AS. ET
5. Pénibles
6. En. Encre
7. Ôtâmes. HEC
8. LED. Kwaï
9. Lee. Entôle
10. Exxon Mobil

I. Boat-people
II. Luirent. Ex
III. Iléon. Alex
IV. Nautisme
V. GNL. Éden
VI. BB. Alès. Nm
VII. Larsen. KTO
VIII. Ite. Schwob
IX. Nove. Reali
X. Gratte-ciel

Nous allons dans le cadre de ce TD/TP continuer notre travail sur les mots croisés et étudier une nouvelle notion très pratique qui est la notion de tableaux dynamiques, c'est-à-dire de tableaux pouvant prendre une taille quelconque et grandir ou diminuer suivant le besoin du programmeur.

Les tableaux que nous avons vus jusqu'à maintenant sont des tableaux statiques, ce qui signifie qu'une fois leur taille fixée, on ne peut jamais la modifier.

Quelques notions sur les tableaux dynamiques

Nous utiliserons l'interface List<E>, que nous avons déjà vu en cours. Comme vous pouvez le constater, il y a un <E>, ce symbole indique que l'on a affaire à une interface générique à laquelle on va passer la **classe** des objets que l'on va stocker dans ce tableau dynamique.

L'implémentation de cette classe peut être faite par plusieurs classes, nous utilisons en général ArrayList<E> mais il est possible d'utiliser d'autres implémentations comme Vector. Par exemple ArrayList<Cercle> indiquera que l'on stockera des objets de type Cercle et ArrayList<Case> des objets de type Case.

ArrayList est une classe ce qui veut dire que pour l'utiliser, on utilisera son constructeur **new ArrayList<Cercle>( )** ; les **( )** indiquant l'appel au constructeur par défaut de la classe.

Comme ArrayList est une des implémentation de l'interface List, on peut utiliser toutes les méthodes définies dans l'interface:

- void add(un objet) : ajoute un objet à la collection (à la fin)
- boolean remove(un objet) : enlève l'objet de la collection
- get(indice) : renvoie l'objet de la collection situé à une certaine position (indice)
- boolean contains(Object o) : retourne vrai si l'objet o est présent dans la collection
- int size() : retourne le nombre d'objets dans la collection
- int clear() pour vider tous les éléments de la liste dynamique

L'exemple suivant

```
import java.util.ArrayList;
import java.util.List;

public class Test {

    public static void main(String[] args){

    List<String> liste = new ArrayList<String>();

   liste.add("première chaîne");
   liste.add(new String("une deuxième chaîne"));

    int i;

    System.out.println("Longueur de la chaîne : " +  (liste.get(0)).length());
        for(i = 0; i < liste.size(); i++)
                System.out.println(liste.get(i));

    }

}
```

– ligne1 : déclare un ArrayList (collection) de chaînes de caractères
– lignes 2-3 : y ajoute 2 objets chaînes de caractères ; remarquer la deuxième forme d'ajout : on
instancie un objet qu'on ajoute directement à la collection ;

– ligne 5 : introduit une boucle à partir de 0, et tant que l'indice est inférieur à la taille du vecteur size()
– ligne 6 : liste.get(i) : renvoie l'élément du tableau liste à la position d'indice i (cet élément est de classe String)
– liste.get(0)).length() : pour l'objet de la classe String, on invoque sa méthode 'length' qui renvoie sa longueur

Vous pouvez constater que pour la création d'une liste, on utilise :
`List<Toto> maListe = new ArrayList<Toto>() ;` pour stocker des éléments de type « Toto » et non pas
`ArrayList<Toto> maListe = new ArrayList<Toto>()` ;

Écriture de la classe Position

Nous souhaitons à présent identifier les emplacements des mots dans la grille, c'est-à-dire les séquences contiguës d'au moins deux cases qui ne sont pas noires. Les positions peuvent être horizontales (lues de gauche à droite) ou verticales (lues de haut en bas). Une position est représentée par une liste de cases (`List<Case>`).

Donnez l'implantation de la classe **Position** à partir des informations suivantes :

- un attribut qui représente les cases (contiguës) de la grille qui composent l'emplacement du mot. Nous utiliserons un attribut `private List<Case> cases` par exemple ;
- une méthode `public String toString()` qui permet d'afficher le mot à cet emplacement, les caractères qui le constituent éventuellement avec des espaces ;
- une méthode `public int taille()` qui rend la taille de l'emplacement de mot (en nombre de caractères ou cases).

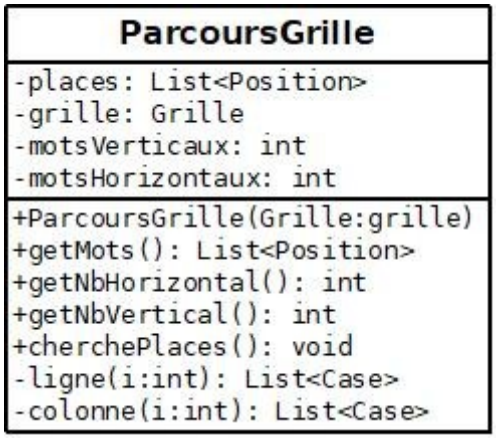
Dans les questions suivantes, nous ajouterons différentes méthodes afin de compléter notre classe.

Écriture de la classe ParcoursGrille

La classe ParcoursGrille doit explorer une Grille pour trouver tous les emplacements des mots qu'elle contient.

Par exemple, dans la grille donnée en introduction de cet énoncé, il y a au total 17 mots horizontaux et 17 mots verticaux.

Écrivez donc la classe ParcoursGrille à partir des informations suivantes :



- donnez le constructeur de cette classe qui prendra une grille en paramètre public ParcoursGrille (Grille grille). Ce constructeur devra entre autre calculez tous les mots que peut contenir cette grille. Nous utiliserons bien évidemment la classe précédente Position pour stocker un mot. La liste de tous les mots possibles sera stockée dans un attribut private List<Position> places. Il faudra donc chercher sur toutes les lignes les emplacements horizontaux puis sur les colonnes les emplacements verticaux. Le processus est expliqué plus loin ;
- une méthode public List<Position> getMots() pour accéder aux mots détectés ;
- une méthode public int getNbHorizontal() pour obtenir le nombre de mots horizontaux ;
- une méthode public String toString() qui permet d'afficher les emplacements de mots détectés ;

Afin de détecter les mots, nous allons procéder de la manière suivante. Nous allons utiliser tout d'abord une méthode List<Case> ligne (int i) qui quand on lui passera une ligne en paramètre retournera la liste des cases de cette ligne.

Par exemple, pour la case (1,1), vous aurez comme résultat :
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10)

Pour la case (2, 1), vous aurez comme résultat :
(2,1) (2,2) (2,3) … (2,10)

et ainsi de suite

La méthode List<Case> colonne (int i) doit faire la même chose pour colonnes cette fois-ci.

En déduire, la méthode private void cherchePlaces() qui cherche les mots dans la liste de cases fournie (une ligne ou une colonne) et qui ajoute les emplacements de mot trouvés à l'attribut **places**. Un emplacement de mot est défini dès que nous avons deux cases contiguës non noires (donc vides ou avec une lettre déjà placée),

Pour écrire cherchePlaces, vous utiliserez bien entendu les méthodes ligne(int i) et colonne(int j). À chaque case rencontrée :

- • si elle n'est pas noire, nous l'ajoutons à la liste des cases;
  sinon, nous examinons la taille de l'emplacement construit :
  - soit il fait plus de deux lettres, c'est effectivement une nouvelle position et on continue et on rajoute cet emplacement ;

sinon on l'oublie

Cette méthode est assez complexe à écrire, prenez votre temps. Vous avez un exemple de fonctionnement ci-dessous :

public class TestMotsCroisés {

```
    public static void main(String [] args){


        Grille g;
        g = ChargeurGrille.lectureGrille("c:\\temp\\grille1.grille");
        System.out.println(g);

        //ChargeurGrille.sauverGrille("c:\\temp\\grille2.grille", g);

        ParcoursGrille pg = new ParcoursGrille(g);

        System.out.println("Affichage des positions de tous les mots de la
grille :");
        System.out.println(pg);

        System.out.println("Nombre de mots horizontaux " +
pg.getNbHorizontal());
        System.out.println("Nombre de mots verticaux " + pg.getNbVertical());
    }
```

Affichage des positions de tous les mots de la grille :
(1,1)(1,2)(1,3)(1,4)(1,5)(1,6)(1,7)(1,8)(1,9)(1,10)
(2,1)(2,2)(2,3)
(2,5)(2,6)(2,7)(2,8)(2,9)(2,10)

(3,1)(3,2)(3,3)(3,4)
(3,6)(3,7)(3,8)
(4,1)(4,2)(4,3)(4,4)(4,5)(4,6)
(4,8)(4,9)(4,10)
(5,5)(5,6)(5,7)
(5,9)(5,10)
(6,1)(6,2)(6,3)(6,4)(6,5)
(6,7)(6,8)(6,9)(6,10)
(7,1)(7,2)(7,3)
(7,5)(7,6)(7,7)(7,8)(7,9)(7,10)
(8,3)(8,4)(8,5)(8,6)(8,7)(8,8)(8,9)(8,10)
(9,1)(9,2)(9,3)
(9,5)(9,6)(9,7)(9,8)(9,9)(9,10)
(10,1)(10,2)(10,3)(10,4)(10,5)(10,6)(10,7)(10,8)(10,9)(10,10)
(1,1)(2,1)(3,1)(4,1)(5,1)(6,1)(7,1)(8,1)(9,1)(10,1)
(1,2)(2,2)(3,2)(4,2)
(6,2)(7,2)
(9,2)(10,2)
(1,3)(2,3)(3,3)(4,3)(5,3)(6,3)(7,3)(8,3)(9,3)(10,3)
(3,4)(4,4)
(1,5)(2,5)
(4,5)(5,5)(6,5)(7,5)(8,5)(9,5)(10,5)
(1,6)(2,6)(3,6)(4,6)(5,6)
(7,6)(8,6)(9,6)(10,6)
(1,7)(2,7)(3,7)
(5,7)(6,7)(7,7)(8,7)(9,7)(10,7)
(1,8)(2,8)(3,8)(4,8)
(6,8)(7,8)(8,8)(9,8)(10,8)
(1,9)(2,9)
(4,9)(5,9)(6,9)(7,9)(8,9)(9,9)(10,9)
(1,10)(2,10)(3,10)(4,10)(5,10)(6,10)(7,10)(8,10)(9,10)(10,10)

Nombre de mots horizontaux 17
Nombre de mots verticaux 17