- Programmation Orientée Objet -

Manipulation de tableaux et de chaînes de caractères

Avant propos

Dans ce TD nous nous intéresserons à la manipulation de texte via l'utilisation de la classe `String` *et à la gestion d'information sous forme de tableaux. Comme dans le TD précédent, nous commencerons par une application en mode Console Nous allons mettre en œuvre différentes notions :*

- *Manipulation de String*
- *Lecture dans un fichier texte*
- *Recherche dans un tableau*
- *Recherche de lettres dans un mot*
- *Interface console*

Contexte

Nous allons nous placer dans le contexte d'un **assistant électronique pour jeux de mots de type Scrabble**. Votre programme devra pouvoir trouver **quels sont les mots du dictionnaire qui peuvent être construits à partir d'un certain nombre de lettres et de jokers** (caractère pouvant correspondre indifféremment à n'importe quelle lettre). Nous voulons aussi pouvoir **trier ces mots par nombre de points rapportés** (sans tenir compte cependant du placement sur le plateau : lettre ou mot compte double ou triple, scrabble, etc.).

Nous allons progresser étape par étape et rajouter des fonctionnalitées au fur et à mesure.

Étape 0 : Classe principale : ScrabbleConsole

EXERCICE 1

Ouvrez un éditeur de texte (`gedit` ou `vim`).
Déclarez une une nouvelle classe publique `ScrabbleConsole` qui contiendra elle-même la méthode `main`.

```
public class ScrabbleConsole {

   public static void main(String[] args) {

   }
}
```

Classe publique En Java, une classe ou une énumération publique DOIT être dans une fichier de MÊME nom.

Fonction main En Java, la méthode `main` possède toujours la même signature :
`public static void main(String[] args)`

Classe principale Cette classe, en interface console, vous servira a tester votre travail au fur et à mesure du TP.

EXERCICE 2

Ajoutez un constructeur sans paramètre à la classe `ScrabbleConsole`. Ce constructeur affichera simplement pour le moment la phrase `"Welcome to the Scrabble assistant "` !.
L'instruction étant dans le constructeur, il va donc falloir créer un nouvel objet `ScrabbleConsole` dans la méthode `main`.

Afficher du texte En Java, la méthode d'affichage la plus courante est :
`System.out.println(String s)`

Compilation Pour compiler, on utilise la commande (dans le Terminal) :
`javac ScrabbleConsole.java`
Pour exécuter, on utilise la commande :
`java ScrabbleConsole`.
Vous devriez voir :

```
Welcome to the Scrabble assistant !
```

Pourquoi mettre le println dans le constructeur ?
Afin de rendre votre classe `ScrabbleConsole` indépendante de la méthode `main`. La méthode `main` est unique. Ce qui est dedans ne peux pas être réutilisé. En mettant vos lignes de codes dans `ScrabbleConsole`, ce dernier pourrait être appelé par un méta-programme ou être réutilisé !

Étape 1 : Le mot juste

Pas de Scrabble sans un dictionnaire ! Notre première classe sera donc la classe `Dictionary`.

Nous désirons que la classe `Dictionary` puisse :

1. **stocker une liste de mots**
2. **répondre à la question : "Cette chaîne de caractère est-elle un mot de la liste ?"**
3. **initialiser la liste de mots à partir d'un fichier texte (ce sera plus simple que de tout rentrer à la main).**

1.1 Stocker une liste de mot

EXERCICE 3

 Créez la classe `Dictionary` puis ajouter lui un attribut **privé** `wordsList` permettant de stocker une liste de mots (un référent sur un tableau de `String` ?).

EXERCICE 4

 Créez un constructeur **public** sans paramètre qui initialise la liste de mot avec les mots suivants : `"abricot"`, `"châtaigne"`, `"groseille"`, `"pomme"`, `"tomate"`.

 **Déclaration d'un objet tableau** En Java, les tableaux sont des objets, ils sont donc alloués dynamiquement. Cela permet de spécifier la taille du tableau au moment de son instanciation. Attention, ils reste de taille fixe durant le programme !

Exemple :

```
String[] myArray; // Declaration
myArray = new String[12]; // Allocation
```

 **Attention !** Comme l'attribut est déclaré dans la classe, mais son initialisation se fait dans le constructeur, la méthode des accolades `int[] tab = {1, 2, 3};` ne marche pas !

EXERCICE 5

 Rajoutez une méthode **publique** `String[] getWordsList()` qui renvoie la liste de mots stockés dans le dictionnaire.

EXERCICE 6

 Modifiez le constructeur de la classe `ScrabbleConsole` pour créer un nouveau `Dictionary` et pour en afficher le contenu !

 **Afficher le contenu d'un tableau** Rappelez-vous : la méthode `Arrays.toString( ... )` permet de convertir un tableau en une chaîne de caractères affichables !

Votre code (côté `ScrabbleConsole`) devrait ressembler à ça :

```
/*
 * Projet d'assistant Scrabble
 */
import java.util.Arrays;
import java.lang.System;

/**
 *
 * @author gademer
 */
public class ScrabbleConsole {

    /* Constructor */
    public ScrabbleConsole() {
        System.out.println("Welcome to the Scrabble assistant !");
        Dictionary dico = new Dictionary();
        System.out.println(Arrays.toString(dico.getWordsList()));
    }

    public static void main(String[] args) {
        ScrabbleConsole sc = new ScrabbleConsole();
    }

}
```

```
Welcome to the Scrabble assistant !
[abricot, châtaigne, groseille, pomme, tomate]
```

1.2 Tester la présence d'un mot

EXERCICE 7


Rajoutez une méthode `isValidWord` à votre classe `Dictionary` qui prend en paramètre une chaîne de caractères et vérifie si cette chaîne est présente dans la liste de mot. Cette méthode renvoie une valeur booléenne (`true`/`false`).

Recherche dans un tableau Algorithme classique de recherche d'un élément :

Initialiser `found` *à faux*
pour chaque élément du tableau **faire**
  si élément courant correspond à l'élément recherché **alors**
    Marquer `found` *comme vrai*
    Sortir de la boucle
  fin
fin
Renvoyer la valeur de `found`

1.3 Initialiser l'objet à partir d'un fichier texte

EXERCICE 8


Récupérez sur http://learning.esiea.fr/ l'archive `fr_FR_utf8.dico.zip` contenant le fichier texte du dictionnaire `fr_FR_utf8.dico`.

Le fichier de dictionnaire fourni est composé :
- du nombre de mots sur la première ligne
- des mots du dictionnaire (un par ligne)
- il contient plus de 350 000 mots !

```
354490
ADSL
ASCII
Abbeville
Abel
Abidjan
Abraham
Abyssinie
Académie
...
```

EXERCICE 9


Rajoutez un **deuxième** constructeur **qui prendra en paramètre une chaîne de caractères** correspondant au nom du fichier dictionnaire **et qui remplira la liste de mots** avec les mots du fichier.


Lire dans un fichier La récupération d'information dans un fichier se fait comme celle des information au clavier par le biais de la classe `Scanner`
http://docs.oracle.com/javase/8/docs/api/index.html?java/util/Scanner.html

Exemple :

```
Scanner scan = new Scanner(new File("monfichier.txt"));

while(scan.hasNext()) {
   System.out.println(scan.next()); // Print the content of the file to the console
}
```

À vous de jouer pour transformer l'exemple précédent pour non pas afficher le contenu, mais le stocker dans le tableau !

En cas d'erreur de lecture du fichier, vous attraperez les exceptions générées et quitterez le programme proprement.

Gestion des erreurs Lorsque vous construisez un nouvel objet `Scanner` **à partir d'un fichier**, celui-ci peut vous renvoyer une exception de type `FileNotFoundException` si le fichier n'existe pas.

Ce n'est pas le rôle du constructeur `Dictionary(String filename)` de gérer cette erreur (ce n'est qu'un maillon de la chaîne), c'est pourquoi on écrit (dans la classe `Dictionary` !) :

```
/* ... */
public Dictionary(String fileName) throws FileNotFoundException {
 /* ... */
}
/* ... */
```

pour indiquer que cette exception sera renvoyée à la méthode appelant le constructeur (méthode dite de la patate chaude ).

En revanche, le constructeur `ScrabbleConsole` appartient à la classe d'interface avec l'utilisateur, c'est donc à elle de "gérer" l'exception.

On écrit (dans la classe `ScrabbleConsole` !) :

```
/* ... */
public ScrabbleConsole() {
 /* ... */
 try { // Filet pour attraper les exceptions
   Dictionary dico = new Dictionary("Petit_Larousse.txt");
 }
 catch(FileNotFoundException ex) { // En cas de FileNotFoundException
   System.err.println(ex.getMessage()); // Afficher un message d'erreur
   System.exit(-1); // Quitter
 }
}
/* ... */
```

Le code précédent devrait vous donner :

```
Welcome to the Scrabble assistant !
Petit_Larousse.txt (Aucun fichier ou dossier de ce type)
```

car le nom de fichier est incorrect.

EXERCICE 10

Modifiez votre classe `ScrabbleConsole` pour afficher :

```
Welcome to the Scrabble assistant !
354490 words loaded. From ADSL to oeuvée.
```

Étape 2 : Des lettres pour former un mot

Au Scrabble, le joueur tire 7 lettres au hasard et doit composer un mot avec ces lettres. On peut alors considérer deux approches :

- soit on recherche les mots du dictionnaire dont toutes les lettres sont contenues dans la liste de lettres piochées.
- soit on génère toutes les permutations possibles de lettres que l'on cherche ensuite dans le dictionnaire.

Quel algorithme ?

Complexité des algorithmes On parle de complexité d'un algorithme pour caractériser son comportement (en temps d'exécution ou en utilisation de la mémoire) lorsque le nombre de données augmente significativement (asymptote à l'infini) ou lorsque le type de données change (cas particuliers). La complexité permet de choisir entre plusieurs algorithmes celui qui est le plus adapté à notre situation.

Pour reprendre l'exemple de l'exercice précédent : une recherche dichotomique évolue globalement en $O(log(n))$, c'est-à-dire que le temps nécessaire pour trouver l'élément est augmente de manière logarithmique par rapport au nombre d'éléments du tableau. En comparaison la recherche linéaire est de complexité linéaire ($O(n)$), ce qui est significativement plus coûteux pour des grands tableaux. Néanmoins sur des tableaux d'une vingtaine d'éléments la recherche linéaire est largement suffisante !

Ici, le nombre de permutation possibles pour n lettres équivaut à $\sum_{k=1}^{n-1} \frac{n!}{(n-k)!}$. Pour n = 7, cela nous donne $\frac{7!}{6!} + \frac{7!}{5!} + \frac{7!}{4!} + \frac{7!}{3!} + \frac{7!}{2!} + \frac{7!}{1!} = 7 + 7 \times 6 + 7 \times 6 \times 5... = 8659$ possibilités !

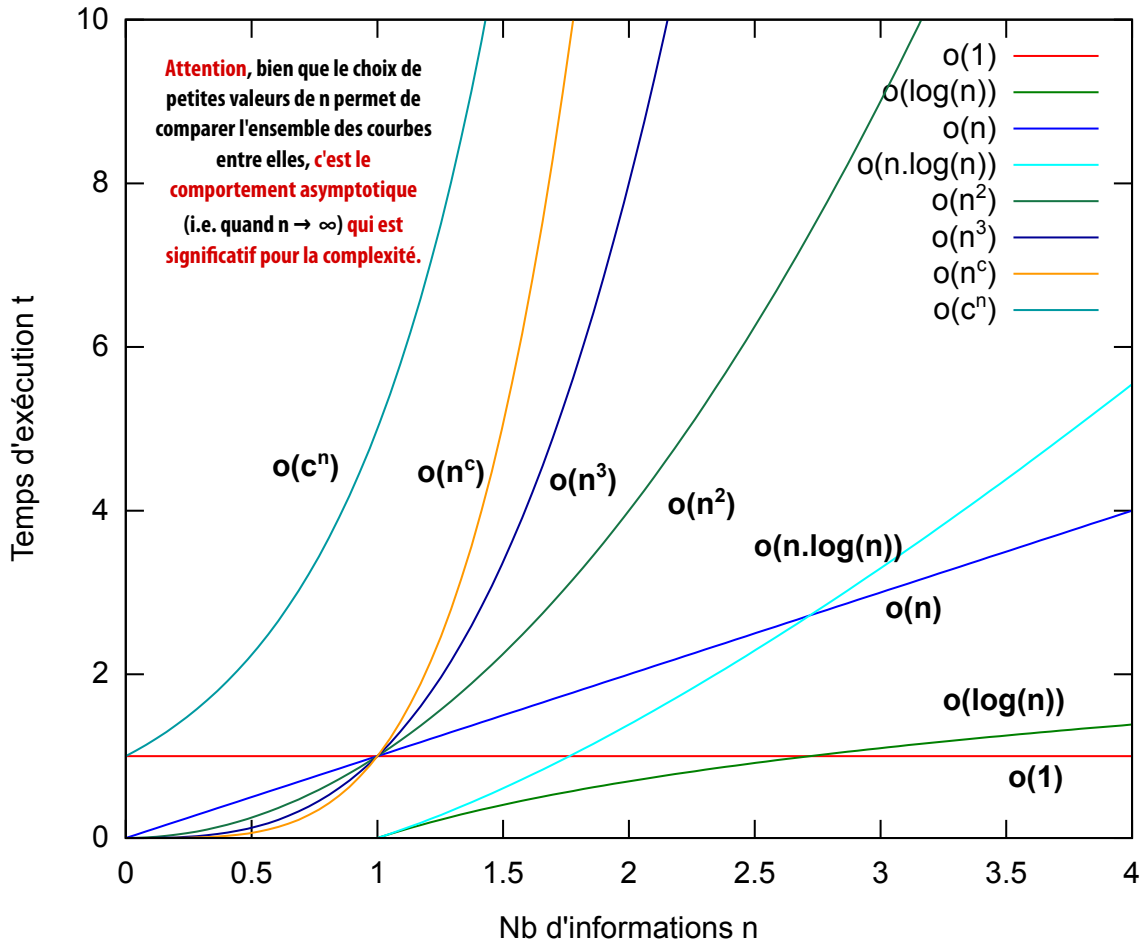
Du coup, on peut en déduire que le comportement de l'algorithme n°2 est exponentiel ($O(e^n)$) et il faut encore multiplier ce temps par la taille du dictionnaire D puisque chaque permutation est recherchée dans le dictionnaire.

Complexité approximative de l'algorithme n°2 : $e^n \times log(D)$ soit $O(e^n)$ si D est constant.

D'un autre côté, le temps de traitement de l'algorithme n°1 dépend de la taille du dictionnaire D, multiplié par le nombre moyen de lettres dans les mots du dictionnaire M, multiplié par le nombre de lettres du joueur.

Complexité approximative de l'algorithme n°1 : $D \times M \times n$, soit $O(n)$ si D et M sont constants

Regardez la figure suivante et faites votre choix :



Est-ce qu'un mot peut être composé avec un ensemble de lettres données ?

Mettons que nous disposions d'une chaine de caractères `String word` (le mot à vérifier) et d'un tableau de caractère `char[] letters` (la liste de lettres disponibles). Nous pouvons par exemple vérifier (par une recherche linéaire) que chacun des caractères de `word` se trouve dans `letters`.

Cependant, il nous faut distinguer les lettres déjà utilisées des lettres encore disponibles.

On peut, par exemple, utiliser un tableau de `boolean` (qui valent `true` ou `false`) dont les indices correspondraient aux indices des caractères de `letters`

Pour remplir un tableau avec une valeur unique (disons `false`) on peut encore utiliser la classe `Arrays`.

```
boolean[] isUsed = new boolean[letters.length] // isUsed is of the same size as letters
Arrays.fill(isUsed, false); // set all elements to false
```

Hypothèse négative

Il est plus simple ici de faire l'hypothèse que vous ne trouverez pas le caractère, puis d'éventuellement invalider l'hypothèse par votre recherche. Notez que vous devrez parcourir toutes les lettres pour être sûr de votre hypothèse mais qu'en revanche vous pouvez quitter la méthode dès que le caractère recherché est dans la liste.

EXERCICE 11

Rajoutez à votre classe `Dictionary` la méthode :

- `public boolean mayBeComposed(String word, char[] letters)` qui renvoie vrai si le mot `word` peut être composé à partir des lettres `letters` et faux sinon (`word` peut avoir moins de lettres que `letters`).

  Puis testez votre fonction.

Affichage probable de votre programme :

```
Welcome to the Scrabble assistant
354490 words loaded. From ADSL to oeuvée.
Please enter a word:
bonjour
Please enter a letter list:
bjoonru
bonjour may be composed with letters : [b, j, o, o, n, r, u]
```

Afficher un tableau La méthode la plus simple consiste à utiliser la méthode `Arrays.toString(Object[] array)` ! de la classe `Arrays` (encore elle ).

Convertir une chaîne de caractères en tableau de caractères : `Scanner` ne vous permet de récupérer qu'un objet `String`. Pour le convertir en tableau de caractères (`char[]`), utilisez la méthode `toCharArray()` de la classe `String`.

Exemple :

```
String myString = "Hello World !";
char[] myArray = myString.toCharArray();
System.out.println("myArray = "+ Arrays.toString(myArray));
```

```
myArray = [H, e, l, l, o, , W, o, r, l, d, , !]
```

Et dans le cas de mot avec des caractères accentués ?

```
Welcome to the Scrabble assistant
354490 words loaded. From ADSL to oeuvée.
Please enter a word:
abbés
Please enter a letter list:
abbes
abbés may NOT be composed with letters : [a, b, b, e, s]
```

Votre programme fait une différence (légitime) entre le e accentué et le e non accentué. Mais au Scrabble on utilise le même jeton voyelle quelque soit l'accent ! De la même manière les caractères œ et æ sont généralement représentés par deux jetons distincts (respectivement o + e et a + e).

EXERCICE 12

Rajoutez à votre classe `Dictionary` la méthode :

- `public static String replaceFrenchCharacter(String s)` qui va effectuer les remplacement suivant :
  - à, â et ä → a,
  - ç → c,
  - é, è, ê et ë → e,
  - î et ï → i,
  - ô et ö → o,
  - ù, ü et û → u,
  - œ → oe,
  - æ → ae.

  La méthode se termine en renvoyant la chaine modifiée (la méthode `replace` de `String` peut être d'un grand secours).

Puis corriger la méthode `mayBeComposed`.

```
Welcome to the Scrabble assistant
354490 words loaded. From ADSL to oeuvée.
Please enter a word:
élève
Please enter a letter list:
eeelv
élève may be composed with letters : [e, e, e, l, v]
```

Majuscule/Minuscule Afin d'éviter les problèmes liés à la casse des lettres, vous pouvez aussi convertir le mot `word` passé en paramètre de la fonction en minuscules avec la méthode de la classe `String` : `toLowerCase()`

```
Welcome to the Scrabble assistant
354490 words loaded. From ADSL to oeuvée.
Please enter a word:
Chine
Please enter a letter list:
cehin
Chine may be composed with letters : [c, e, h, i, n]
```

Static or not static

Si une méthode n'a pas besoin d'instancier un objet (i.e. elle n'a pas besoin des attributs), alors elle peut être déclarée `static`.

L'intérêt est de pouvoir utiliser cette méthode sans avoir besoin d'un objet. Par exemple, dans notre classe `Dictionnary`, un appel du type `Dictionary.replaceFrenchCharacter(letters)` entraine l'application de la méthode `replaceFranchCharacter` sur la `String letters` alors même qu'il n'y a pas eu d'instanciation d'un objet de type `Dictionary`.

Recherche sur tout le dictionnaire Maintenant que nous sommes capable de vérifier si un mot peut être formé à partir des lettres choisies, il ne nous reste plus qu'à appliquer la vérification sur tous les mots du dictionnaire !

EXERCICE 13

Rajoutez la méthode `public String[] getWordsThatCanBeComposed(char[] letters)` qui retourne un tableau contenant tous les mots du dictionnaire pouvant être écrit à partir de la liste de lettre `letters`.

Puis modifier la classe `ScrabbleConsole` pour tester votre programme !

Notez ici que l'on ne connaît pas, à l'avance, la taille de la liste de mots retenus par votre méthode. Il vous faut donc faire deux passes : une première pour savoir combien de cases réserver, et une deuxième pour remplir le tableau. Nous verrons plus tard comment créer des Collections et des tableaux dynamiques.

À ce stade l'affichage de votre programme devrait ressembler à :

```
Welcome to Scrabble assistant
354490 words loaded. From ADSL to oeuvée
Please enter a letter list:
machine
115 matching words found : [Aimé, Ain, Caen, Caïn, Chine, Cie, Han, Mach, Maine, Manche, Me,
Nice, a, ace, ache, acmé, acné, aiche, aie, aime, aimé, ainé, ami, amie, aminé, anche, anime,
animé, aîné, aï, came, camé, cane, chai, chaine, chainé, chaîne, chaîné, chemin, chemina,
chia, chie, chien, china, chine, chiné, chié, cime, ciné, cinéma, e, eh, en, h, ha, haie,
haine, han, haï,haï, haïe, haïe, hein, hem, hi, hic, hie, hé, i, in, inca, m, ma, mac, mach,
machin, machine, machiné,mai, maie, man, manche, mancie, manie, manié, me, mec, men, mena,
menai, mi, mica, miche, mie, mien,min, mina, mince, mine, miné, mâche, mâché, mécha, méchai,
Ça, Çà, à, âme, âne, ça, çà, échina, écima,émia, émiça]
```

Étape 3 : Le Joker

Notre programme remplit une partie du cahier des charges : nous pouvons rechercher des mots du dictionnaire à partir d'une liste de lettres. Mais au Scrabble, certains jetons servent de joker : ils peuvent être utilisés pour représenter n'importe quelle lettre !

Comment gérer les joker dans notre recherche de mot ?

Considérons le comportement de notre programme (`'*'` est utilisé ici comme caractère joker) :

```
Please enter a letter list:
bec
1 matching word(s) found : [bec]
Please enter a letter list:
be*
0 matching word(s) found : []
```

Notre méthode `mayBeComposed` a refusé le mot `"bec"` car il n'a pas pu trouver la lettre `'c'`.

Comment changer le comportement de `mayBeComposed` pour gérer les jokers figurés par le caractère `'*'` ?

Une solution compliquée consiste à essayer d'utiliser des expressions régulières pour définir une notion d'équivalence partielle...

Indice : Une solution simple est de considérer que les jokers nous autorisent à rater notre recherche de lettre 1 fois par joker !

EXERCICE 14

Modifiez la méthode `mayBeComposed` pour gérer l'utilisation de jokers.

Avec un joker on récupère plein d'autres mots possibles !

```
Welcome to Scrabble assistant
354490 words loaded. From ADSL to oeuvée
Please enter a letter list:
be*
61 matching words found : [Me, Web, Xe, a, ab, bd, be, bec, bel, ben, ber, bey, blé, br, bu,
bue, bye, béa, bée, d, de, dé, e, eh, en, es, et, eu, ex, f, g, h, hé, i, j, je, k, lb, le,
lé, m, me, o, p, q, r, ré, t, te, té, u, v, w, web, x, y, z, à, ès, éd, ô]
```

Étape 4 : Gestion du score

Quelle score pour un mot ? Pour avoir le nombre de points associé à un mot il nous faut connaître les lettres utilisées pour le composer. En effet, nous ne devons pas compter les jokers utilisés qui valent 0 points. Cela devrait être facile en repartant de la méthode `mayBeComposed`, vu que nous avons le tableau `isUsed` a notre disposition !.

EXERCICE 15

 En vous basant sur la méthode `mayBeComposed`, créez la méthode :
`public static char[] getComposition(String word, char[] letters)` dans la classe `Dictionary` qui renvoie les lettres effectivement utilisées pour composer le mot sous la forme d'un nouveau tableau de `char`. Pour cela il suffit de compter dans le tableau `isUsed` combien de caractères sont utilisés, puis de les affecter au tableau. On rajoute à ce tableau autant de jokers que de caractères manquants. (En cas de composition impossible renvoyez `null` à la place de `false`.)

EXERCICE 16

 **Créer un nouveau fichier `ScrabbleComparator.java`** pour une nouvelle classe `ScrabbleComparator` qui possède les méthodes :

- `public int letterValue(char letter)` qui renvoie la valeur d'une lettre dans les règles du Scrabble Français.
- `public int lettersValue(char[] letters)` qui renvoie la valeur d'un ensemble de lettres.

Afin de pouvoir récupérer la composition des mots, nous allons ajouter un attribut et un constructeur à notre classe :

- `private` char[] letters
- `public ScrabbleComparator(char[] letters)`

En conséquence, nous allons pouvoir créer la méthode `public int wordValue(String word)` (nécessitant d'instancier la classe) qui renvoie le nombre de points pour un mot donné (le jeu de lettres étant spécifié à la construction du `ScrabbleComparator` et en faisant un bon usage de la méthode `getComposition`).

Au Scrabble Français les lettres ont les valeurs suivantes :

- joker → 0 point
- e, a, i, n, o, r, s, t, u, l → 1 point
- d, m, g → 2 points
- b, c, p → 3 points
- f, h, v → 4 points
- j, q → 8 points
- k, w, x, y, z → 10 points

Quelles sont les mots qui rapportent le plus ? Au Scrabble chaque lettre est associée à une valeur en points (en lien avec la rareté de la lettre). Nous aimerions afficher d'abord les mots qui rapportent le plus !

Mais la méthode `Arrays.sort(Object[] a)` ne fait le tri que selon l'ordre alphabétique... à moins de lui fournir notre propre relation d'ordre (plus petit/plus grand) sous la forme d'une implémentation de l'interface `Comparator<String>` !

http://docs.oracle.com/javase/8/docs/api/index.html?java/util/Comparator.html

EXERCICE 17

 Modifiez `ScrabbleComparator` pour qu'elle implémente l'interface `Comparator<String>`.

La méthode `public int compare(String s1, String s2)` doit renvoyer :
- **-1 si on veut que s1 soit rangé avant s2,**
- **0 si elles ont la même valeur et**
- **1 si on veut que s2 soit rangé avant s1.**

Utilisation du comparateur :

```
String[] wordList = dic.getWordsThatCanBeComposed(letters);
ScrabbleComparator sc = new ScrabbleComparator(letters);
Arrays.sort(wordList, sc);
```

```
Welcome to Scrabble assistant
354490 words loaded. From ADSL to oeuvée
Please enter a letter list:
115 matching words found : [chemina, machine, machiné, Manche, chemin, machin, manche, méchai,
chaine, chainé, chaîne, chaîné, miche, mâche, mâché, mécha, échina, Chine, Mach, aiche, anche,
chien, china, chine, chiné, mach, ache, chai, chia, chie, chié, cinéma, mancie, émința, haine,
hic, mince, écima, acmé, came, camé, cime, haie, haïe, haïe, hein, hem, mica, Caen, Caïn, Han,
Maine, Nice, acné, aminé, anime, animé, cane, ciné, han, haï, haï, hie, inca, mac, manie,
manié, mec, menai, Aimé, Cie, ace, aime, aimé, amie, eh, ha, hi, hé, maie, mena, mien, mina,
mine, miné, émia, ainé, ami, aîné, h, mai, man, men, mie, min, Ça, Çà, âme, ça, çà, Ain, Me,
aie, ma, me, mi, âne, aï, en, in, m, a, e, i, à]
```

Avec quelques améliorations nous obtenons l'interface ci-dessous :

```
Welcome to Scrabble assistant
354490 words loaded. From ADSL to oeuvée
Please enter a letter list:
be*
61 matching words found : [Web ([b, e, *] 4), be ([b, e, *] 4), bec ([b, e, *] 4), bel ([b,
e, *] 4), ben ([b, e, *] 4), ber ([b, e, *] 4), bey ([b, e, *] 4), blé ([b, e, *] 4), bue ([b,
e, *] 4), bye ([b, e, *] 4), béa ([b, e, *] 4), bée ([b, e, *] 4), web ([b, e, *] 4), ab ([b,
*] 3), bd ([b, *] 3), br ([b, *] 3), bu ([b, *] 3), lb ([b, *] 3), Me ([e, *] 1), Xe ([e, *]
1), de ([e, *] 1), dé ([e, *] 1), e ([e, *] 1), eh ([e, *] 1), en ([e, *] 1), es ([e, *] 1),
et ([e, *] 1), eu ([e, *] 1), ex ([e, *] 1), hé ([e, *] 1), je ([e, *] 1), le ([e, *] 1), lé
([e, *] 1), me ([e, *] 1), ré ([e, *] 1), te ([e, *] 1), té ([e, *] 1), ès ([e, *] 1), éd ([e,
*] 1), a ([*] 0), d ([*] 0), f ([*] 0), g ([*] 0), h ([*] 0), i ([*] 0), j ([*] 0), k ([*] 0),
m ([*] 0), o ([*] 0), p ([*] 0), q ([*] 0), r ([*] 0), t ([*] 0), u ([*] 0), v ([*] 0), w ([*]
0), x ([*] 0), y ([*] 0), z ([*] 0), à ([*] 0), ô ([*] 0)]
```

Étape 5 : Mini-interface graphique

Jusqu'ici nous avons interagi avec nos classes uniquement par le biais de la console. Essayons de créer une interface simple qui permet de récupérer la liste de lettres et d'afficher le résultat.

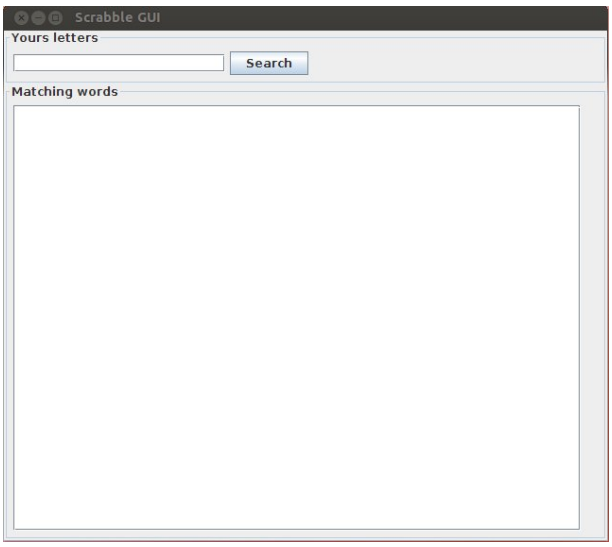
EXERCICE 18

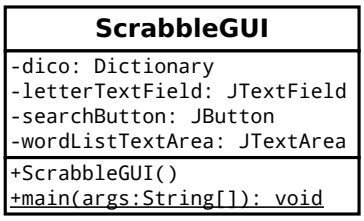
Créez une classe `ScrabbleGUI` :
- qui crée une interface composée de deux zones : en haut un panneau avec un champ de saisie `JTextField` et un bouton `JButton`, et en dessous un panneau avec un champ d'affichage texte (`JTextArea`),
- qui charge le dictionnaire et affiche le nombre de mots chargés dans la console.

La classe `ScrabbleGUI` possède aussi une méthode `public static void main(String[] args)` qui se contente de créer un nouvel objet `ScrabbleGUI`.

http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JFrame.html
http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JPanel.html
http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JButton.html http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JTextField.html http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JTextArea.html

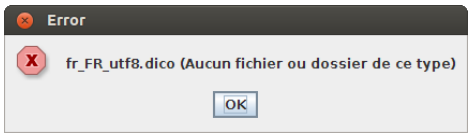
Pour définir une taille par défaut d'un élément graphique, on utilise la méthode `setMinimuSize`.





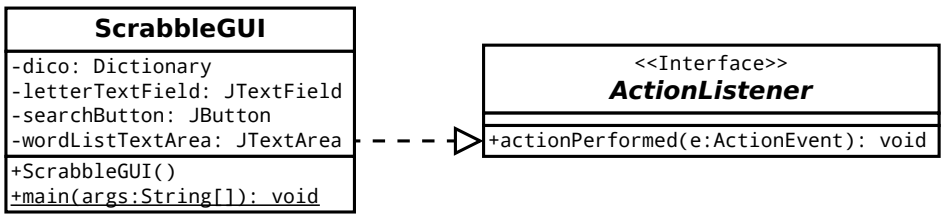
En cas d'exception vous pouvez afficher une fenêtre de dialogue d'erreur avec la commande suivante :

```
try { /* ... */ } catch (Exception ex) {
   JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
   System.exit(-1);
}
```



EXERCICE 19

Modifiez `ScrabbleGUI` pour quelle implémente l'interface `ActionListener` et reliez le bouton à la classe.
Modifiez la méthode `public void actionPerformed(ActionEvent e)` pour que l'appui du bouton provoque la lecture du champ de saisie (`JTextField`), le calcul des mots que l'on peut composer puis l'affichage dans la zone d'affichage (`JTextArea`). En cas de doute, référez-vous au TD précédent.

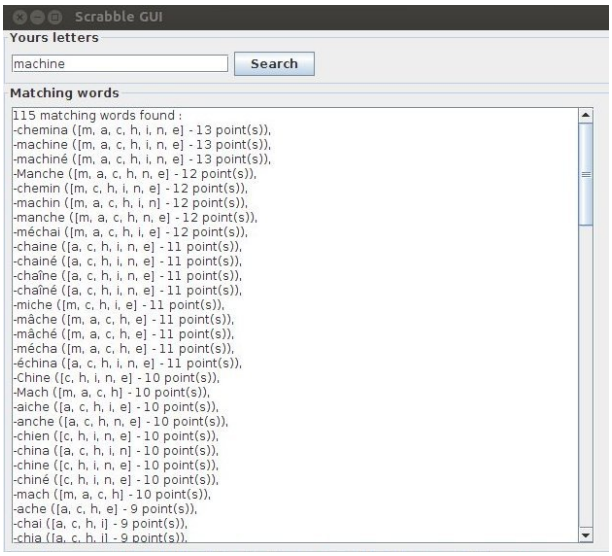


Zone d'affichage texte avec ascenseurs Lorsque le texte à afficher dépasse la zone visible, on utilise généralement une zone à défilement de type `JScrollPane`.
http://docs.oracle.com/javase/7/docs/api/index.html?javax/swing/JScrollPane.html

Exemple d'utilisation de `JScrollPane`

```
JTextArea wordListTextArea = new JTextArea();
JScrollPane scroll = new JScrollPane(wordListTextArea);
panel.add(scroll);
```

Vous devriez alors obtenir une interface proche de :



Bravo ! Vous avez désormais votre programme avec deux interfaces possibles : console ou graphique !

The end. 😄

Pour aller plus loin

Parce qu'on peut toujours avoir d'autres envies, on peut envisager :
- la possibilité de rajouter une ou des lettres correspondant aux "points d'accroche", c'est-à-dire aux lettres déjà sur le plateau et auxquellles le mot du joueur peut se raccrocher. On considéra dans un premier temps que l'on ne peut utiliser que l'une des lettres d'accroche à la fois.
- rajouter la spatialisation avec la position des mots dans l'espace, les bonus compte double ou triple et les mots déjà déposés !
- Ce dont vous avez envie !