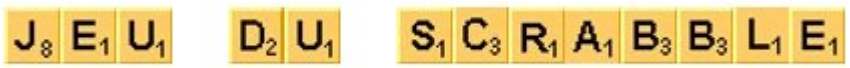
*Algorithmique et structures de données*

# TD / TP 9

C. Sempé

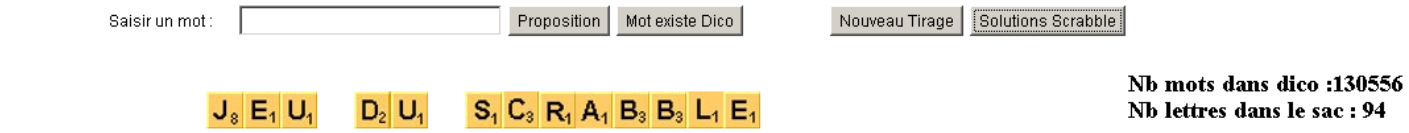
Les objectifs de cette séance de TD / TP sont :

- Utiliser les Arbres Binaires de recherche
- Utiliser la programmation récursive
- Utiliser les structures ArrayList, LinkedList et HashMap.




## TD

Nous voulons mettre en œuvre le jeu ci-dessous :




- Les lettres du joueur sont tirées au hasard dans un sac.
- D'autre part, nous avons un dictionnaire avec tous les mots possibles du Scrabble.
- L'idée dans un premier temps est de pouvoir proposer des solutions au joueur.
- Puis, lui indiquer si un scrabble est possible, vérifier si sa proposition est acceptable ... on peut inventer le jeu que l'on veut ...

## Le dictionnaire

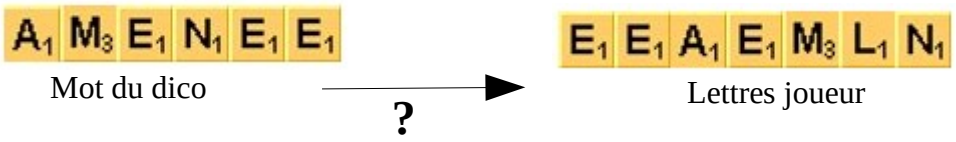
Nous allons utiliser un arbre binaire de recherche comme dictionnaire. On souhaite charger environ 130000 mots. Une classe **Mot** stockera chaque mot du dictionnaire.

```
public class Mot
{
private String LeMot;

Mot( String UnMot )
public String getMot()

public boolean equals( Mot o )
public int comparer( Mot o)

public String toString()
```

```
private boolean lettresPresentes(String s)
private boolean uneLettrePresente( char a)
private int nbLettres( char a, String s )
public boolean motOK( String s)
```

Le mécanisme de base consiste à comparer un Mot du dico avec un ensemble de lettre.

La question centrale est : Puis-je constituer ce mot du dico avec cet ensemble de lettre ?




## 1.Recherche d'une lettre

- Ecrire une méthode itérative qui indique si une lettre donnée en argument est dans une chaine s :
  **`private boolean lettrePresente(char a, String s)`**

(Vous utiliserez la méthode charAt(i) de la classe String qui retourne le caractère de la chaîne qui se situe au rang i.)

## 2. Nombre d'occurrences

- Ecrivez une méthode itérative qui permet de déterminer le nombre d'occurrences de la lettre **a** dans le mot **s**.
  **`private int nbLettres( char a, String s )`**

## 3.Recherche des lettres d'un mot dans une chaine ( inclusion)

- Ecrivez une méthode itérative **lettresPresentes (String s)** qui indique si toutes les lettres du MOT sont bien dans la chaîne s ( paramètre donné à la méthode)

  **`private boolean lettresPresentes(String s)`**

## 4.Validation du mot

- Ecrivez la méthode motOK(String s) qui vérifie que le mot contenu dans l'objet est bien inclus dans la chaîne donnée en paramètre et que le nombre d'occurrences de chacune des lettres du mot de l'objet est bien inférieur ou égal au nombre d'occurrences de la chaîne s.
  `public boolean motOK( String s){}`

## 5.Méthodes de comparaison

- Ecrire la méthode **comparer(Mot o)** qui permet de comparer 2 mots. On utilisera l'ordre alphabétique comme relation d'ordre entre les mots. Vous donnerez la réponse 0 si les deux mots contiennent la même chaîne de caractères, -1 si l'objet est alphabétiquement placé avant le mot **o**, +1 si inverse.

- Ecrire la méthode **equals(Mot o)** qui indique si les deux mots sont égaux

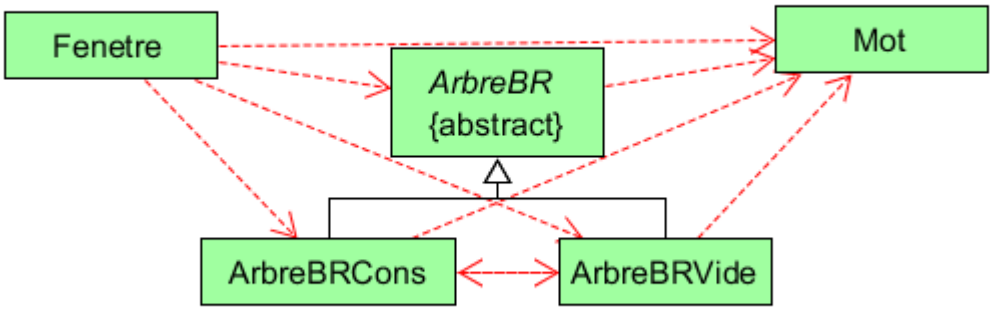
**TP**

## 6.Test classe Mot

- Ouvrez le fichier Mot.java
- Complétez la classe Mot puis faites les tests unitaires nécessaires pour contrôler le bon fonctionnement des méthodes motOK ...

```
…
Mot m1 = new Mot("PASSERA") ;
if( m1.lettresPresentes("SSAAPERZZ")
…
```

- Ouvrez les autres fichiers




- 

Compilez et exécutez

Observez les variables de la classe Fenetre et identifiez bien le dictionnaire, les lettres du joueur, le tableau possibilité qui stockages mot intéressants.
Le chargement du dico, sac, points est fait dans les structures de données respectives. Identifiez-les.

L'attribut **this.possibilites** de la classe Fenêtre est un ArrayList de Mot qui contiendra toutes les solutions retenues afin de les afficher.

## 7.Des solutions (*bouton Solutions Scrabble*)

Nous voulons collectionner dans **this.possibilites** tous les mots qui vont correspondre à la combinaison du joueur **this.lettresJoueur**.

Modifiez la classe ArbreBR et faites la méthode :
**rechercheSolutions (String lettresJoueur, ArrayList <Mot> possibilites)**

A chaque fois que le mot du dictionnaire est motOk() alors il faut l'ajouter à possibilités
Limitez les mots trouvés à 7 caractères afin de limiter les solutions.

Faites appel a cette méthode en donnant les bons paramètres dans la classe Fenetre dans la methode actionPerformed, lorsqu'on appuis sur le bouton "Solutions Scrabble".

Posez vous la question : mon parcours doit il utiliser la relation d'ordre de ABR ou non ?

## 8.Conserver les solutions optimales avec comme critère leur longueur

Il faut écrire une méthode **elagueParLongueur()**. Cette méthode a pour objet de parcourir l'ensemble des solutions trouvées au point précédent et de ne conserver que les solutions les plus longues du tableau.

On partira des solutions stockées dans le tableau Possibilités au point précédent
Vous utiliserez un tableau ArrayList intermédiaire dans lequel vous stockerez les résultats.
En fin de méthode, vous copierez ce tableau intermédiaire dans la variable Possibilités.
Posez vous la question : si un scrabble est possible, est-il dans le tableau ?

## 9.Gérer la proposition de l'utilisateur.

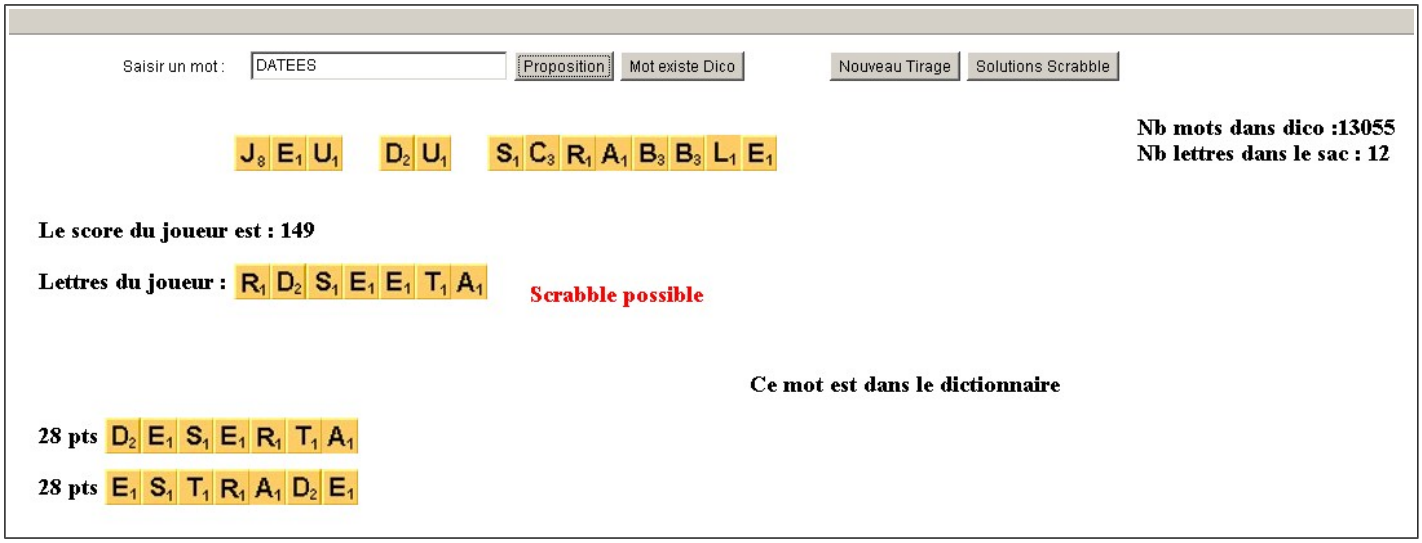
La zone de saisie permet de saisir la proposition de l'utilisateur. Vous placerez la valeur saisie dans la zone dans la variable **proposition**. Ce dernier validera sa proposition en appuyant sur le bouton correspondant.
Le mot proposé par l'utilisateur.(Proposition) doit être conforme au tirage, doit exister dans le dictionnaire.
En conséquence, dans la zone de traitement du bouton 'Proposition', il faut :

- récupérer le texte saisi;
- Le transformer en majuscules;
- recherche le mot dans le dictionnaire. ( si c'est OK, placer la variable **motPossible** à true)
- S'il existe, on regarde si le mot est conforme au tirage.
- Si les deux conditions précédentes sont vérifiées,
  - on calcule le score du mot proposé.
  - on le cumule avec le score déjà existant dans la variable **score**
  - on affiche les solutions ( comme avec le bouton Solutions)

Amélioration : on pourra calculer le score de l'ordinateur en même temps pour le comparer à celui du joueur.

## 10.Bouton <Mot existeDico>

Le bouton <Proposition> doit rechercher dans le dictionnaire le mot proposé par l'utilisateur saisi dans la zone de texte. Si ce mot est valide, alors afficher son score, l'ajouter au score déjà obtenu, afficher les solutions de l'ordinateur.
Le bouton <existeDico> donne simplement l'indication de présence du mot saisi dans le dictionnaire.
Les solutions scrabble affichent les solutions de l'ordinateur.




## 11.Construire les solutions avec comme critère les meilleurs scores.

## 12.Afficher le message 'Scrabble possible' si le tirage le permet.