1. Préalable

Vous devez implémenter les méthodes d'une classe et répondre à 2 questions posées à même le fichier de la classe.

1.1. Préparation

- Le projet est un projet Java standard, SANS BESOIN D'UTILISER MAVEN.
- Créez un projet Java de base sous votre IDE.
- Téléchargez et décompressez le package fourni sur umtice.
- Rappatriez le package présent sur umtice dans votre projet, à la racine de ce dernier (dossier src/) : ne le mettez pas dans un autre package, pour éviter d'avoir des problèmes de compilation.
- Le package contient a minima :
    - une interface exprimant les méthodes à implémenter, avec leur javadoc ;
    - une classe principale implémentant cette interface, et présentant les éléments suivant déjà implémenté :
        - tous les attributs nécessaires ;
        - les constructeurs nécessaire ;
        - possiblement certaines méthodes privées utilitaires ;
        - une méthode main et des méthodes de test pour tester vos implémentation (à partir de la méthode main, vous n'avez pas besoin de lire ou comprendre tout le code) ;
        - **un cartouche de commentaire contenant 2 questions auxquelles vous devrez répondre.**
    - Selon le sujet, le package peut contenir d'autres classes manipulées par la classe principale.

1.2. Travail à Faire

- Lire le sujet du projet ci-dessous
- Implémenter les méthodes dans la classe qui n'ont le sont pas déjà. **ATTENTION** : la javadoc fournie dans l'interface (recopiée dans la classe) est la source primaire de règle à respecter, veillez à bien la suivre.
- Vous pouvez exécuter à tout moment la classe pour déclencher une série de test. **ATTENTION** : les tests ne couvrent pas forcément tous les cas possibles.
- Répondre directement dans le fichier de la classe aux questions posées en commentaire dans le cartouche en amont de la définition de la classe.

VOUS NE DEVEZ IMPLEMENTER QUE LES MÉTHODES DE L'INTERFACE DANS LA CLASSE. Ces méthodes sont déclarées dans la classe mais ne sont pas encore implémentée (elle lèvent l'exception de type *UnsupportedOperationException*)

1.3. Rendu :

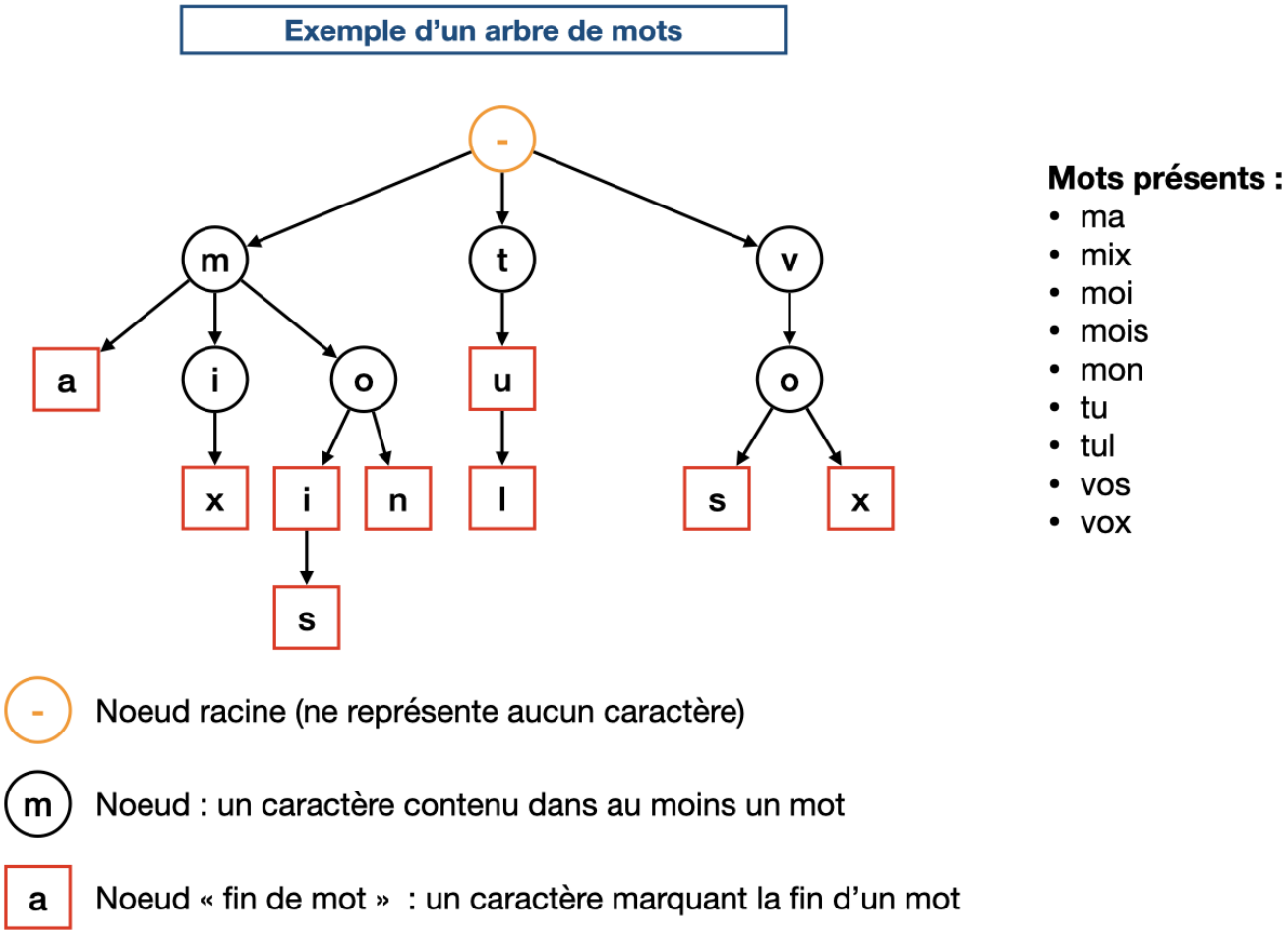
- Déposez sur umtice **uniquement** le fichier .java de la classe dont vous avez implémenté les méthodes et répondu aux questions.
- Ne compressez pas ce fichier dans une archive, n'ajoutez aucun autre fichier.
- Le non respect de ces règles entrainera automatiquement la note de 0.

2. Projet : le correcteur orthographique

Vous devez implémenter un correcteur orthographique simple, qui permet de stocker ou de retirer des mots et d'en vérifier l'existence. Plutôt que de stocker les mots sous forme d'une liste, nous souhaitons stocker les mots au travers d'un arbre de caractère. Chaque caractère est un nœud qui possède une forêt de caractères suivant, possiblement vide. Bien entendu, dans une forêt, un nœud d'un caractère donné n'est présent au plus qu'une fois. Un nœud qui termine un mot (le dernier caractère d'un mot) possède un marqueur de fin de mot.

Un arbre ayant forcément une racine unique, le premier nœud de l'arbre ne représente aucun caractère particulier et n'est présent que pour rattacher au sein de sa forêt les premiers caractères de chaque mot connus du correcteur.

Voici une illustration d'un tel arbre :



Pour les 5 méthode à implémenter, il possible d'avoir une approche itérative ou récursive, bien que l'approche récursive soit recomandée pour cet examen, et que la classe CorrecteurOrthoImpl soit prévue à cette effet.

Sont exposés brievement ci-dessous les algorithmes à employer pour l'ajout et le retrait d'un mot.

2.1. ajout récursif d'un mot

Le principe de l'ajout récursif d'un mot est d'avoir une méthode récursive qui prend en paramètre le *mot* et l'indice *i* du caractère courant à ajouter.

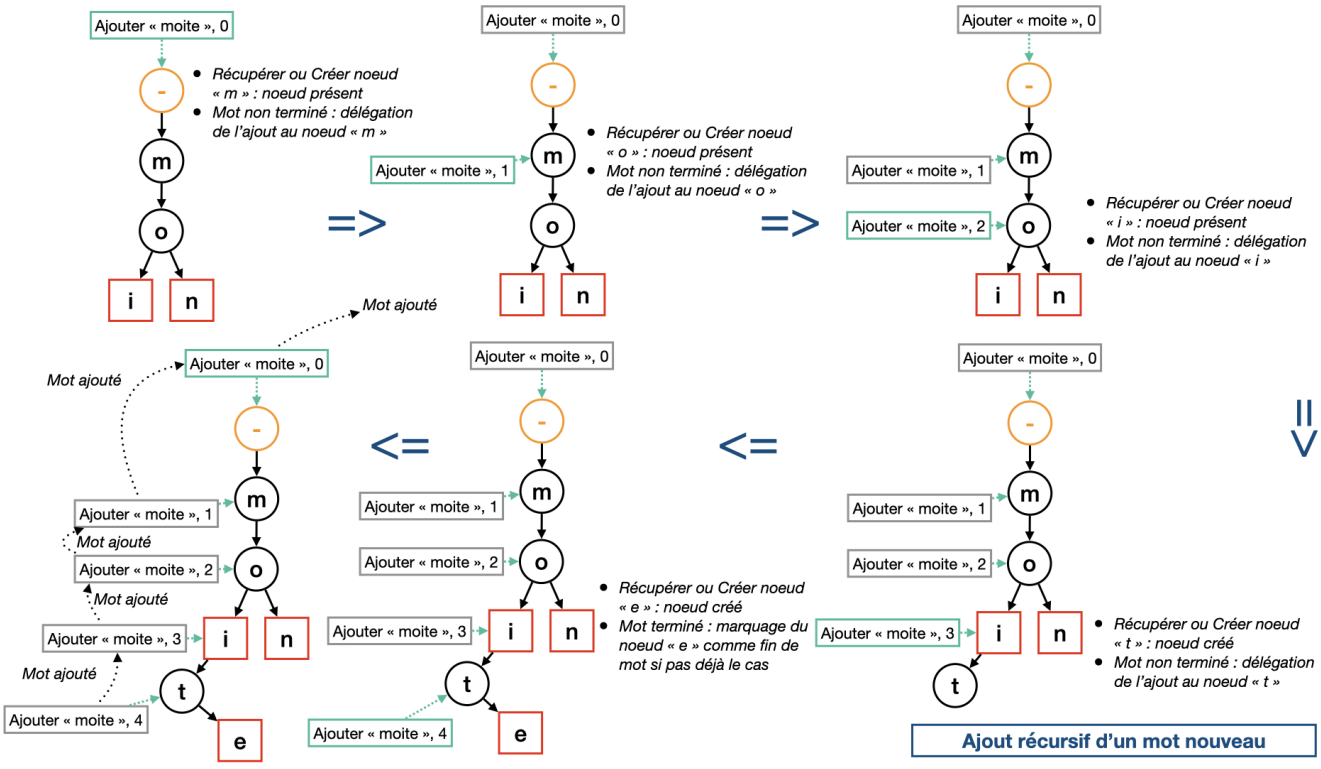
L'appel de cette méthode sur un arbre de caractères a pour étapes principales :

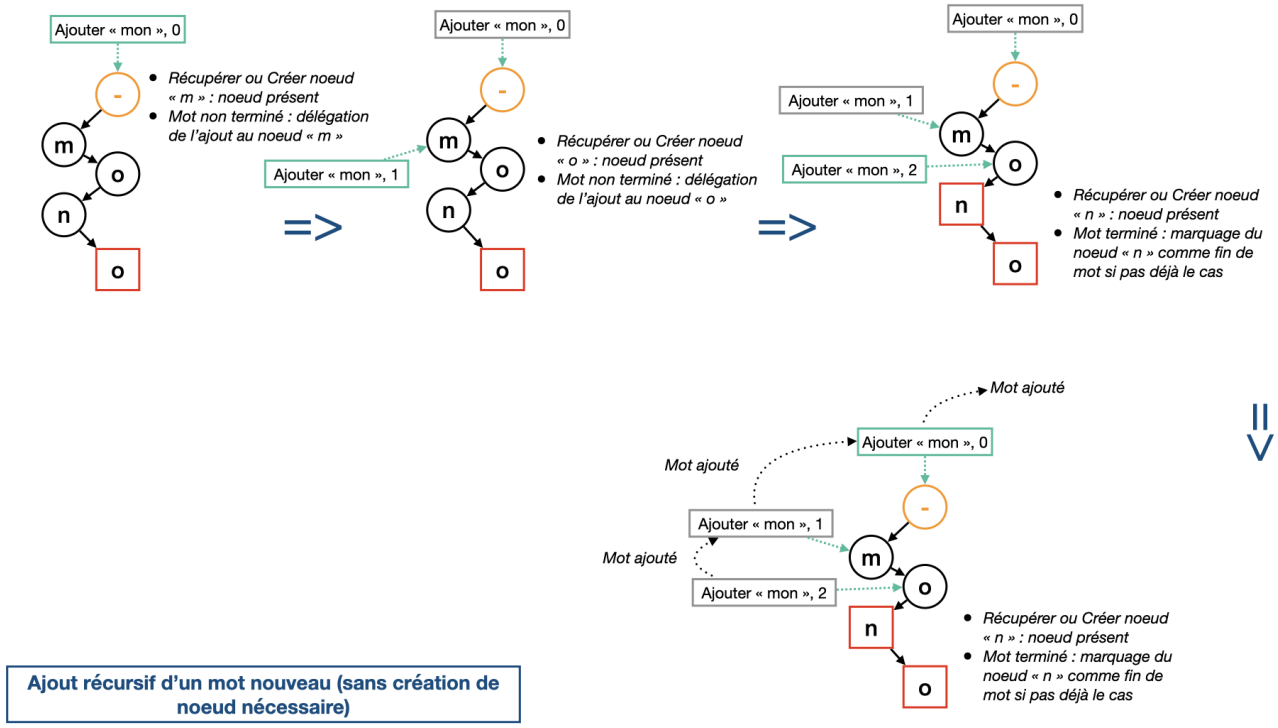
- récupérer ou ajouter dans sa forêt l'arbre *A* dont la racine correspond au caractère *i* du *mot* ;
- si le caractère *i* n'est pas le dernier :
    - invoquer la méthode récursive d'ajout sur *A* pour le *mot* et l'indice *i+1* et retourner son résultat (mot ajouté ou non);
    - sinon si _A _n'a pas de marqueur de fin de mot :
        - positionner le marqueur de fin de mot sur *A* et retourner le fait que le mot ait été ajouté ;
    - sinon (A a déjà un marqueur de fin de mot):
        - retourner le fait que le mot n'ait pas été ajouté (déjà existant).

Des illustrations de l'ajout de mots sont données à la fin de cette section.

La méthode publique *ajouterMot(String mot)* n'est pas récursive en l'état. Celle ci sert donc uniquement à :

- vérifier les préconditions : mot non nul, non vide et non "blanc" (constitué que de caractères blancs : espace, tabulation, saut de ligne) ;
- "corriger" le mot : supprimer les possibles caractères blancs en début et fin de mot et le passer en minuscule.
- invoquer la méthode récursive d'ajout avec comme mot le mot "corrigé" et comme indice celui du premier caractère du mot.





Ajout récursif d'un mot nouveau (sans création de noeud nécessaire)

2.2. retrait récursif d'un mot

Le principe du retrait récursif d'un mot est d'avoir une méthode récursive qui prend en paramètre le *mot* et l'indice *i* du caractère courant du mot. Si le mot est bien présent dans l'arbre, celui-ci est "retiré" : le nœud du dernier caractère voit son indicateur de fin de mot retiré (positionné à faux). Ensuite, pour éviter que l'arbre ne grossissent en mémoire indéfinimment, en remontant les nœuds des caractères du mot, ces derniers sont supprimés s'ils sont devenus inutile : s'il ne sont pas des fin de mots et que leur forêt est vide.

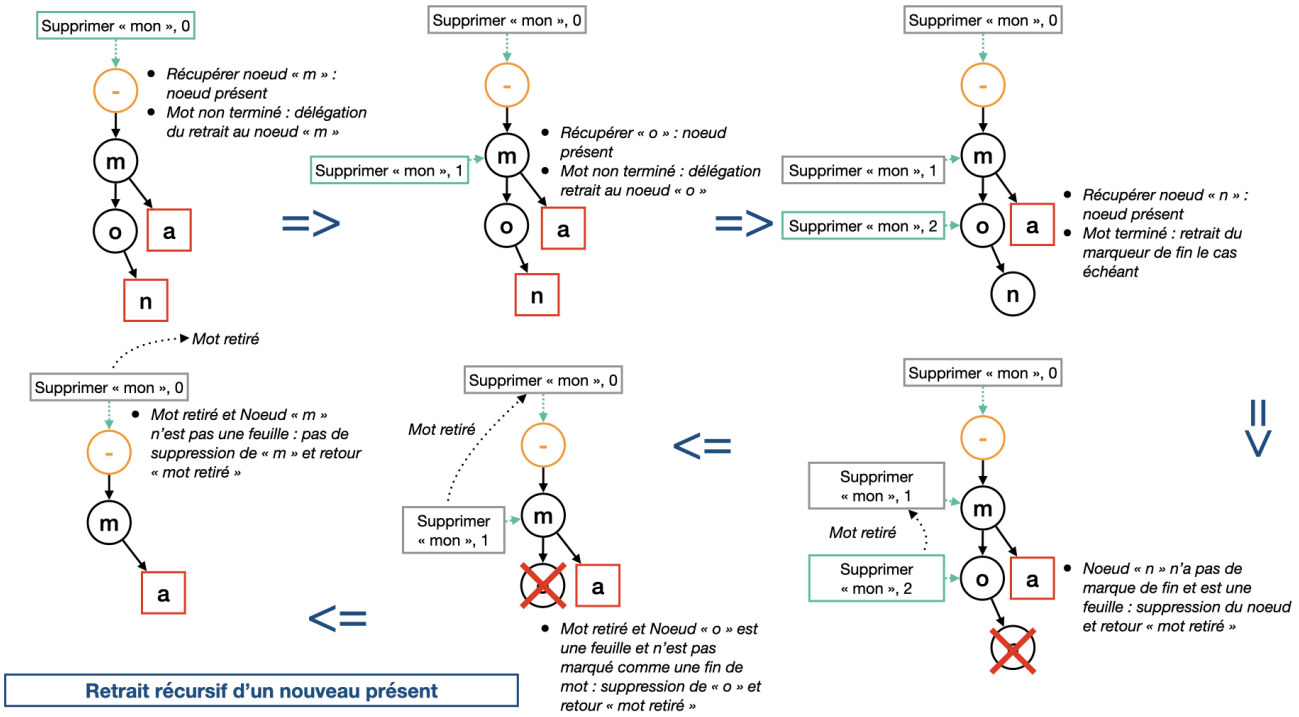
L'appel de cette méthode sur un arbre de caractère a pour étapes principales :

- *Parcours récursif du mot dans l'arbre pour retrait :*
  - récupérer dans la forêt de l'arbre courant l'arbre *A* dont la racine correspond au caractère *i* du *mot* ;
  - si l'arbre *A* n'existe pas :
    - s'arrêter là et retourner le fait que le mot n'ait pas été supprimé (mot absent) ;
  - sinon, si le caractère *i* n'est pas le dernier :
    - invoquer la méthode récursive de retrait sur l'arbre *A* pour le *mot* et l'indice *i+1* ;
    - si la méthode invoquée retourne le fait que le mot n'ait pas été supprimé :
      - s'arrêter là et retourner le fait que le mot n'ait pas été supprimé (mot absent) ;
  - sinon si l'arbre *A* n'a pas le marqueur de fin de mot :
    - s'arrêter là et retourner le fait que le mot n'ait pas été supprimé (mot absent) ;
  - sinon :
    - retirer l'indicateur de fin de mot de l'arbre *A* (le positonner à faux) ;
- *Nettoyage (suppression des nœuds devenus inutiles) :*
  - si l'arbre A n'a pas d'indicateur de fin de mot, et que l'arbre A est une feuille (n'a pas de sous-arbre) :
    - supprimer l'arbre A de la forêt de l'arbre courant ;
  - retourner le fait que le mot ait été supprimé.

Une illustration du retrait d'un mot est donnée à la fin de cette section.

La méthode publique *retirerMot(String mot)* n'est pas récursive en l'état. Celle ci sert donc uniquement à :

- vérifier les préconditions : mot non nul, non vide et non "blanc" (constitué que de caractères blancs : espace, tabulation, saut de ligne) ;
- "corriger" le mot : supprimer les possibles caractères blancs en début et fin de mot et le passer en minuscule.
- invoquer la méthode récursive de retrait d'un mot avec comme mot le mot "corrigé" et comme indice celui du premier caractère du mot.



2.3. Tips :

- Plusieurs méthodes de la classe String pourraient vous être utiles : *empty*, *isBlank*, *trim*.

3. Questions (reportées sur la classe java, réponses à apporter directement dessus)

Question 1

Sachant que la création d'une `ArrayList`, même vide, provoque la réservation en mémoire de 10 emplacements par défaut, quelle amélioration proposeriez-vous pour gagner de l'espace en mémoire dans la gestion du correcteur ?

Question 2

Un autre développeur à choisi de réaliser son correcteur en utilisant une table de Hachage `HashMap<String, Boolean>` où les mots sont stockés comme clé et la valeur booléene n'est pas utilisée

(mise à vrai par défaut). Il prétend obtenir de meilleurs "performances" qu'avec votre système. Selon vous, quels sont les avantages et inconvénients de sa proposition par rapport à la votre ?