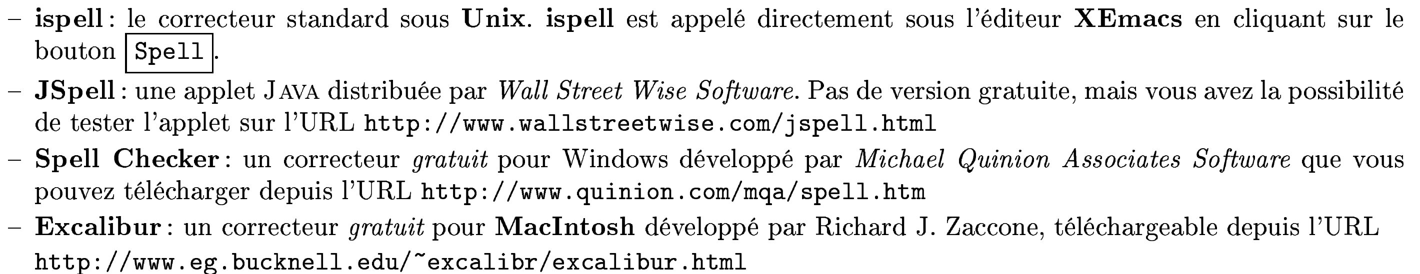
JEs

Le but de ce projet est de réaliser un *correcteur orthographique* pour l'*anglais* baptisé `JEs` (`Java English spellchecker`). Ce projet s'inspire d'un correcteur orthographique déjà ancien appelé `epelle`[1].

La partie 1 définit et présente le fonctionnement général d'un correcteur orthographique. La partie 2 développe en détail les fonctionnalités de `JEs` que vous devez réaliser en tout ou partie. La partie 3 détaille certains points concernant la réalisation de `JEs`. La partie 4 propose une architecture possible pour le correcteur. La partie 5 propose un planning de travail pour la réalisation du projet. La partie 6 explique quand et comment vous devez remettre votre travail.

Si vous voulez vous familiariser avec la manipulation d'un correcteur orthographique, vous pouvez essayer :



- **ispell** : le correcteur standard sous **Unix**. **ispell** est appelé directement sous l'éditeur **XEmacs** en cliquant sur le bouton `Spell`.
- **JSpell** : une applet JAVA distribuée par *Wall Street Wise Software*. Pas de version gratuite, mais vous avez la possibilité de tester l'applet sur l'URL `http://www.wallstreetwise.com/jspell.html`
- **Spell Checker** : un correcteur *gratuit* pour Windows développé par *Michael Quinion Associates Software* que vous pouvez télécharger depuis l'URL `http://www.quinion.com/mqa/spell.htm`
- **Excalibur** : un correcteur *gratuit* pour **MacIntosh** développé par Richard J. Zaccone, téléchargeable depuis l'URL `http://www.eg.bucknell.edu/~excalibr/excalibur.html`

1 Présentation

Un correcteur orthographique est un programme qui tente de *détecter* et de *corriger* les fautes d'orthographe dans un texte. On entend ici par *faute d'orthographe* une erreur lexicale, c'est à dire un mot mal orthographié, à l'exclusion de tout autre type de faute (faute d'accord, de conjugaison, de genre, etc). Ainsi, la phrase "`A phew Miss steaks aye can knot sea`" est orthographiquement correcte en anglais, car elle ne contient que des mots anglais corrects.

La *détection* d'erreur lexicale consiste à s'assurer que chaque mot du texte à corriger correspond à une forme réellement existante dans un langage donné. Cette forme peut être un mot que l'on trouve dans un dictionnaire classique, mais aussi les variations possibles de ce mot en genre et en nombre, ainsi que toutes les conjugaisons si ce mot est un verbe. Si le mot ne correspond à aucune forme connue, alors il est *inconnu*.

La *correction* d'erreur lexicale consiste à proposer à l'utilisateur une liste de mots corrects possibles pouvant éventuellement se substituer au mot inconnu. Par exemple, pour le mot `tble`, le correcteur anglais peut proposer les mots de remplacement `table`, `able`, `tale` ou `tile`.

Dans un correcteur orthographique, l'ensemble des mots connus sont stockés dans un ou plusieurs *dictionnaires*. Un dictionnaire est simplement un ensemble de mots, comme l'ensemble des mots anglais usuels. Le correcteur peut utiliser des dictionnaires relatifs à des domaines particuliers, comme par exemple les termes informatiques. Un dictionnaire est représenté par une structure de données en général non triviale, la simple liste de mots (même triée alphabétiquement) n'offrant pas des performances suffisantes tant sur le plan du temps d'exécution que sur l'espace mémoire utilisé. Certains correcteurs utilisent des techniques basées sur le *hachage* (comme **ispell**), d'autres (comme `JEs`) manipulent des ensembles de mots implémentés par des arbres.

Un mot peut être absent du dictionnaire dans lequel le correcteur le cherche mais être néanmoins un mot correct. Dans ce cas, l'utilisateur a la possibilité d'*ajouter* ce mot au dictionnaire en question. Une autre possibilité est de considérer un mot inconnu comme correct seulement durant la correction du texte en cours, et de l'insérer dans un dictionnaire temporaire initialement vide. Un correcteur orthographique est un programme exclusivement *interactif* et la détection d'un mot inconnu entraîne un choix (interactif) de l'utilisateur parmi les possibilités évoquées précédemment.

Un correcteur orthographique est en général associé à un *éditeur*, comme par exemple `ispell` et `Emacs`, ou encore le correcteur de l'éditeur **Word** de Microsoft. Le correcteur vérifie la mémoire tampon de l'éditeur et remplace les mots incorrects. Un correcteur peut également vérifier les mots contenus dans un fichier de texte, sans pour autant éditer ce fichier. Dans les deux cas, le texte ne contient pas que des mots, mais aussi des caractères de ponctuation ou des caractères spéciaux. Le correcteur doit alors *extraire* les mots contenus dans la mémoire tampon de l'éditeur ou dans le fichier, en ignorant les caractères qui ne font pas partie des mots.

---

1. *Epelle : un logiciel de détection de faute d'orthographe*, Paul Zimmermann, rapport de recherche INRIA numéro 2030, Septembre 1993.

2 Fonctionnalités

JEs est un programme interactif qui détecte et corrige des mots mal orthographiés (i.e. des chaînes de caractères) dans un texte. Les mots vérifiés doivent suivre les règles lexicales de l'*anglais*. Le texte en question peut se réduire au seul mot dont on veut vérifier l'orthographe, ou bien consister en un ensemble de mots contenus dans une chaîne de caractères ou dans un fichier. JEs utilise des dictionnaires qui sont des ensembles de mots stockés dans une structure de données particulière (un arbre lexical). Un dictionnaire peut-être stocké sur disque et chargé dans JEs dynamiquement.

2.1 Les dictionnaires

JEs utilise des *dictionnaires*. On entend ici par *dictionnaire* un ensemble de mots qui suivent certaines règles lexicales. Les dictionnaires principaux contiennent les mots anglais usuels. JEs peut aussi utiliser des dictionnaires spécialisés comme par exemple un dictionnaire des termes informatiques en anglais. Un dictionnaire possède une certaine représentation en mémoire et une autre représentation sur disque, où il est stocké sous la forme d'un fichier de texte d'un format particulier (voir 3.1). Les opérations relatives aux dictionnaires sont :

- la **création** : JEs doit être capable de créer des dictionnaires à partir de simple fichiers de mots.
- le **chargement** : étant donné un dictionnaire stocké sur le disque (soit un fichier de texte d'un format particulier), il doit pouvoir être *chargé* dans JEs. On doit pouvoir également **créer** et chager un *nouveau* dictionnaire, initialement vide.
- l'**insertion** : on doit pouvoir insérer un nouveau mot dans un dictionnaire chargé. Une fois inséré, le mot est connu.
- la **sauvegarde** : un dictionnaire chargé peut être modifié durant la correction d'un texte, à la suite de l'insertion d'un nouveau mot. On doit pouvoir *sauvegarder* ce dictionnaire chargé sur le disque de deux façons différentes :
  - en produisant une nouvelle version du fichier de sauvegarde.
  - en produisant un fichier de mots (des chaînes de caractères) contenant tous les mots de ce dictionnaire (c'est l'inverse de la **création**).

JEs connaît initialement deux dictionnaires pour l'anglais, un dictionnaire contenant les mots usuels et un dictionnaire contenant des noms propres et des sigles.

2.2 Le correcteur

Initialement, JEs ne contient aucun dictionnaire. L'utilisateur peut **charger** un ou plusieurs dictionnaires dynamiquement (par exemple le dictionnaire *"english"* pour le vocabulaire anglais courant et le dictionnaire *"computer"* pour les termes informatiques). Le correcteur **cherche** chaque mot à vérifier dans chacun des dictionnaires chargés. Si le mot existe dans un de ces dictionnaires, il est connu. Si le mot n'existe dans aucun de ces dictionnaires, il est inconnu. Lorsque le mot est inconnu, le correcteur doit **proposer** à l'utilisateur une liste de mots *proches* du mot inconnu. L'utilisateur peut alors effectuer un certain nombre d'actions qui dépendent du *mode*.

JEs utilise un algorithme simple pour trouver les mots proches d'un mot inconnu : dans chaque dictionnaire chargé, JEs cherche tous les mots (connus) qui ne diffèrent que d'*une seule lettre* du mot inconnu. Un mot Y ne diffère d'un mot X que d'une seule lettre si on peut passer de X à Y :

- en enlevant une lettre quelconque à X
- en ajoutant une lettre quelconque à X
- en remplaçant une lettre d'X par une lettre quelconque
- en permuttant deux lettres quelconques d'X *consécutives*

Par exemple, le mot blid n'est pas un mot anglais connu, et JEs peut proposer les mots blind, bid, lid, bled ou blip pour la substitution, mais pas les mots bide, blade, bride, elide, glide ou slide.

2.3 Les modes

JEs vérifie l'orthographe des mots issus de diverses *sources*. On peut vérifier l'orthographe d'un mot unique. Le mot est saisi dans une zone *"texte"* de l'interface et simplement vérifié. On peut vérifier l'orthographe de tous les mots d'un texte saisi dans un **mini-éditeur**. Dans ce cas, JEs vérifie chaque mot du texte et remplace éventuellement les mots inconnus par d'autres mots. Enfin, JEs peut vérifier l'orthographe des mots contenu dans un fichier.

2.3.1 le mode mot

L'utilisateur saisit un mot dont il veut vérifier l'orthographe. Si le mot est inconnu, `JEs` **propose** une liste de mots *proches* du mot inconnu. Ces mots proches sont cherchés dans tous les dictionnaires chargés au moment de la vérification. L'utilisateur peut s'il le désire **insérer** le mot inconnu dans un des dictionnaires chargés.

2.3.2 le mode texte

L'utilisateur saisit un texte dans un **mini-éditeur** et `JEs` vérifie chaque mot contenu dans la mémoire tampon de l'éditeur. Pour chaque mot inconnu, `JEs` **propose** une liste de mots *proches* cherchés dans les dictionnaires chargés. L'utilisateur peut alors :

- **ignorer** l'erreur : le mot incorrect est simplement ignoré, et la mémoire tampon de l'éditeur est inchangée.
- **ignorer** l'erreur pour **tout le reste de la session** : le mot incorrect est simplement ignoré, la mémoire tampon de l'éditeur est inchangée, et le mot est maintenant *connu* durant la correction du texte restant. Le mot n'est pas pour autant chargé dans un des dictionnaires chargés, et lors d'une autre correction, il sera de nouveau inconnu.
- **choisir** un des mots proposés par `JEs` : ce mot est alors **substitué** au mot inconnu dans le **mini-éditeur**.
- **corriger** lui-même le mot inconnu : le mot corrigé est alors **substitué** au mot inconnu dans le **mini-éditeur**. L'utilisateur peut s'il le désire **insérer** le mot corrigé dans un des dictionnaires chargés.
- **insérer** le mot inconnu dans un des dictionnaires chargés : la mémoire tempon du **mini-éditeur** reste inchangée. Du fait de l'insertion du mot dans un des dictionnaires chargés, ce mot est désormais connu pour le reste de la correction.

Durant la correction, la mémoire tampon du **mini-éditeur** n'est pas modifiable par l'utilisateur.

2.3.3 le mode fichier

L'utilisateur choisit un fichier de texte et `JEs` vérifie chaque mot contenu dans le fichier. Pour chaque mot inconnu, `JEs` se comporte comme dans le mode **mot**.

2.3.4 le mode fichier/fichier

L'utilisateur choisit un fichier de texte et `JEs` *recopie* le contenu *intégral* de ce fichier dans un fichier temporaire, en vérifiant au passage l'orthographe des mots. Pour chaque mot inconnu, `JEs` **propose** une liste de mots *proches* cherchés dans les dictionnaires chargés. L'utilisateur peut alors :

- **ignorer** le mot inconnu : le mot incorrect est simplement ignoré et il est recopié dans le fichier temporaire.
- **ignorer** le mot inconu pour **tout le reste de la session** : le mot incorrect est simplement ignoré, il est recopié dans le fichier temporaire et il est maintenant *connu* durant la correction de la partie restante du fichier. Le mot n'est pas pour autant chargé dans un des dictionnaires chargés, et lors d'une autre correction, il sera de nouveau inconnu.
- **choisir** un des mots proposés par `JEs` : ce mot est alors **substitué** au mot inconnu dans le fichier temporaire.
- **corriger** lui-même le mot inconnu : le mot corrigé est alors **substitué** au mot inconnu dans le fichier temporaire. L'utilisateur peut s'il le désire **insérer** le mot corrigé dans un des dictionnaires chargés.
- **insérer** le mot inconnu dans un des dictionnaires chargés : le mot inconnu est alors *copié* dans le fichier temporaire.

A l'issue de la recopie/vérification, `JEs` propose à l'utilisateur de sauvegarder le fichier temporaire, éventuellement dans le fichier original.

2.4 Le format *"texte"*

Dans le mode **mot**, `JEs` vérifie l'orthographe d'un mot, c'est à dire d'une chaîne de caractères. Le mot est saisi dans une zone *"texte"* sous la forme d'une chaîne de caractères et seuls les éventuels espaces en début et en fin de chaîne sont ignorés.

Dans les modes **texte**, **fichier** et **fichier/fichier**, `JEs` vérifie un texte au format *"texte"*, c'est à dire une alternance de mots (des chaînes de caractères) et de caractères séparateurs (les caractères de ponctuation, les espaces, etc). Par exemple, si le texte (contenu dans la mémoire tampon du **mini-éditeur** ou dans un fichier) est

```
You␣know␣what??␣␣␣I'm...␣happy!!
```

`JEs` extrait et vérifie les mots `You`, `know`, `what`, `I'm` et `happy` (le caractère ␣ représente l'espace). Remarquez qu'on considère le caractère `'` comme une lettre (voir 3.3). Notez que dans le mode **fichier/fichier**, `JEs` *recopie* directement dans le fichier temporaire tout ce qui n'est pas un mot à vérifier, comme par exemple la suite de caractères `??␣␣␣`.

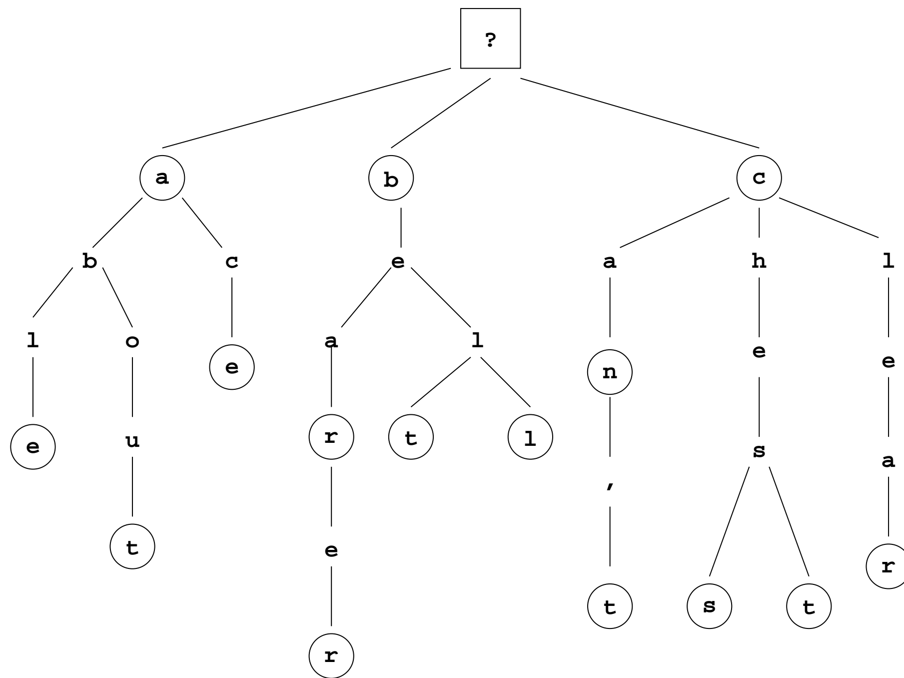


FIG. 1 – *Un arbre lexical n-aire*

2.5 L'environnement initial

Au lancement de `JEs`, un ou plusieurs dictionnaires peuvent être chargés automatiquement (un correcteur sans dictionnaire n'est pas très utile). Pour ce faire, `JEs` lit un *fichier de configuration* qui contient le nom du ou des dictionnaires à charger. Ce fichier possède un nom particulier et réside dans un répertoire particulier. Si entre deux sessions, un dictionnaire mentionné dans le fichier de configuration n'est plus accessible, il est simplement ignoré et `JEs` ne provoque pas d'erreur.

3 Mise en œuvre

On décrit ici en détail certains points de la réalisation de `JEs`. Vous devez implémenter tout ou partie de ce qui vous est présenté en respectant les consignes, quand il y en a. En particulier, l'utilisation des arbres lexicaux est **obligatoire**. Afin de faciliter la correction de votre travail, le *format* de sauvegarde de ces arbres (voir 3.1) est également **imposé**.

3.1 Les arbres lexicaux

La structure de données principale du correcteur est le dictionnaire qui contient un ensemble de mots connus. Il existe différentes techniques pour représenter un dictionnaire, comme par exemple le hachage. Nous choisissons pour `JEs` une technique basée sur les *arbres lexicaux*. Un arbre lexical est un arbre n-aire dont les nœuds contiennent des caractères. Un mot est stocké dans l'arbre suivant un chemin qui mène de la racine vers un nœud *terminal*, c'est à dire un nœud qui correspond à la dernière lettre d'un mot. Par exemple, les mots :

`a b c able about ace bear bearer belt bell can can’t chess chest clear`

peuvent être stockés dans l'arbre lexical de la figure 1. Le caractère stocké à la racine de l'arbre est indéfini et on le considère comme le caractère vide. Un chemin allant de la racine à un nœud de l'arbre est le préfixe d'au moins un mot du dictionnaire. Les nœuds cerclés représentent des fins de mots. Les arbres lexicaux sont aussi appelés *arbres de préfixes* car ils font apparaître les préfixes communs aux mots stockés. Remarquez que le caractère `’` apparaît dans l'arbre comme une lettre.

Une implémentation naturelle d'un arbre n-aire en JAVA utilise deux classes, une classe pour les nœuds de l'arbre et une classe pour les listes de sous-arbres. Par exemple, l'arbre lexical contenant les mots `able`, `about` et `ace` est représenté par la figure 2. Le coût en nombre d'*objets* de cette implémentation est trop élevé pour un arbre lexical réaliste qui risque de contenir un grand nombre de nœuds. Une solution classique à ce problème consiste à représenter l'arbre n-aire à l'aide d'un arbre *binaire*. L'arbre de la figure 2 devient celui de la figure 3. Dans un arbre binaire vu comme un arbre n-aire, le sous-arbre gauche d'un nœud représente le premier fils et le sous-arbre droit un frère, ce qui est mis en évidence sur le dessin de droite de la figure 3 dans lequel les liens vers les sous-arbres gauches sont verticaux et les liens vers les sous-arbres droits horizontaux.

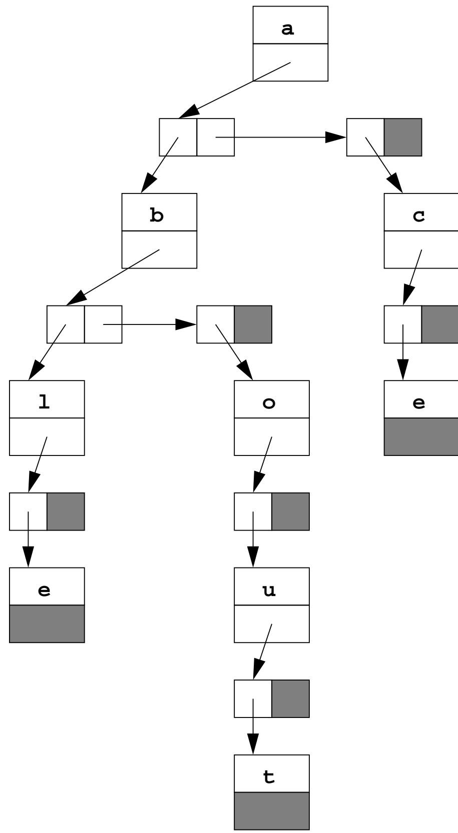


Fig. 2 – *Un arbre n-aire en* Java

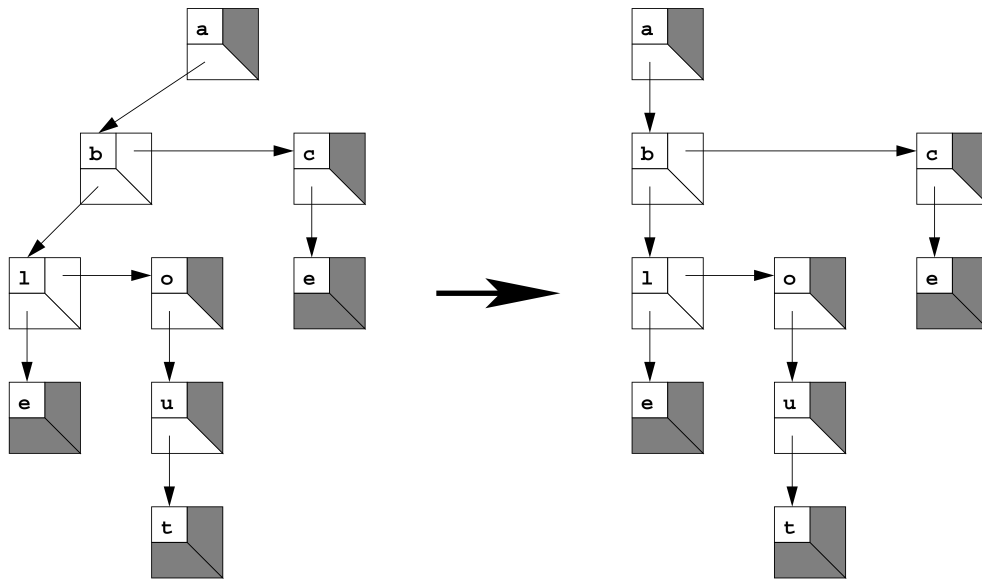


Fig. 3 – *Un arbre n-aire représenté par un arbre binaire*

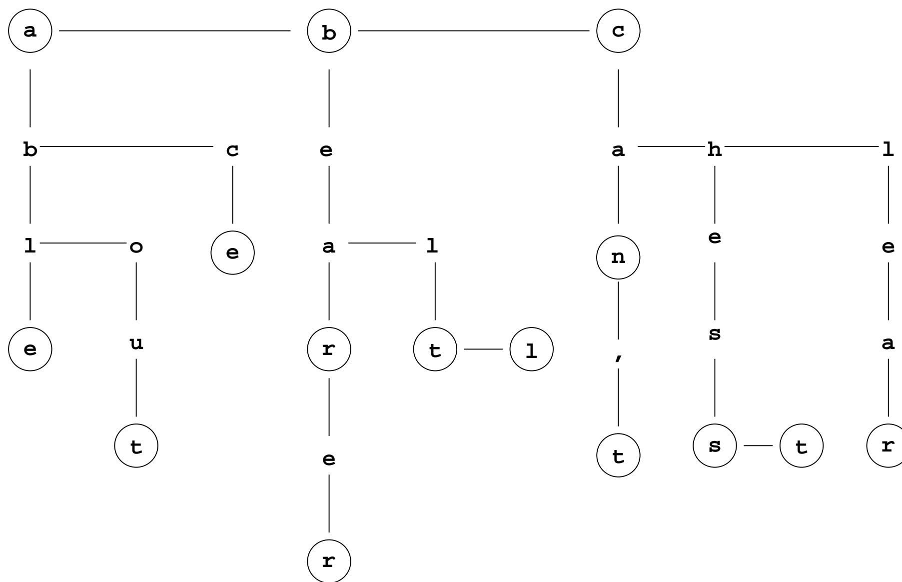


Fig. 4 – *Une fratrie lexicale binaire*

L'arbre binaire correspondant à l'arbre lexical de la figure 1 est celui de la figure 4. Notez au passage qu'on peut se passer de la racine contenant la lettre indéterminée ? et manipuler une *fratrie*, c'est à dire un arbre ayant des *frères*. Remarquez qu'afin de réaliser un gain en espace appréciable, on ne représente *pas* l'arbre vide par un objet mais simplement par la valeur `null` (les champs grisés dans les figures). Ce choix a des conséquences sur l'écriture des méthodes opérants sur ces arbres. En particulier, l'objet *arbre lexical vide* n'existe pas et lors du parcours d'un arbre lexical, il faut tester l'existence de ses sous-arbres *avant* de leur appliquer des méthodes.

Un arbre lexical étant la structure de données choisie pour implémenter un dictionnaire, on doit pouvoir stocker un tel arbre sur disque. On peut facilement stocker un arbre lexical binaire dans un fichier de texte et le relire (ou du moins le reconstruire) depuis ce même fichier. L'écriture d'un arbre dans un fichier s'appelle la *linéarisation*. Les informations à écrire ou à lire sur et depuis le fichier sont des caractères représentant les nœuds de l'arbre, plus éventuellement des informations particulières comme une fin de mot ou une feuille. On peut par exemple linéariser l'arbre de la figure 4, ce qui donne la suite de caractères suivante :

```
abc/ahl/e/a/r*e/s/s$t*/n$/’/t*e/al/t$l*/r$/e/r*bc/e*lo/u/t*e*
```

Devinez les règles qui permettent de linéariser un arbre lexical de cette façon et utilisez ce codage pour écrire et relire vos arbres lexicaux sur et depuis un fichier de texte. Notez que vous devez vous conformer **strictement** à ce codage, et les fichiers de sauvegarde produits par votre correcteur ne doivent contenir **aucune** autre information (commentaires, caractères spéciaux, lignes vides, etc).

3.2 Les dictionnaires

Un dictionnaire `JEs` est un arbre lexical plus d'autres informations, comme par exemple le *nom* du dictionnaire. Un dictionnaire existe dans l'application sous la forme d'un arbre lexical et existe sur le disque sous la forme d'un fichier de texte qui est l'arbre lexical correspondant linéarisé suivant la méthode expliquée précédemment. Pour les distinguer des autres, ces fichiers doivent posséder une extension spéciale, par exemple le suffixe `.jes`. Ces fichiers doivent tous être placés dans le même répertoire. Le chemin spécifiant ce répertoire peut être passé en paramètre lors du lancement de l'application. Il semble raisonnable d'utiliser un nom identique pour un dictionnaire vu depuis (l'interface de) l'application et pour le fichier qui le représente sur disque. Par exemple, le dictionnaire de nom `english` est sauvegardé dans le fichier `english.jes`.

Initialement, `JEs` ne connait *aucun* dictionnaire (c'est à dire qu'il n'existe aucun fichier avec le suffixe `.jes` dans le répertoire). Par ailleurs, `JEs` doit être capable de créer des dictionnaires à partir de simple fichiers de mots. Autrement dit, il faut pouvoir créer un fichier `.jes` à partir d'un fichier `.txt` (ici, l'extension n'est qu'indicative) contenant une suite de mots (à raison d'un mot par ligne). Un dictionnaire *en mémoire* pouvant être modifié par l'*insertion* d'un nouveau mot, il faut pouvoir effectuer l'opération inverse, c'est à dire produire un fichier de mots à partir du dictionnaire *en mémoire* (c'est à dire à partir de l'arbre lexical).

Deux fichiers de mots sont fournis, les fichiers `english.txt` et `english-proper.txt`, contenants respectivement une liste de mots anglais usuels et une liste de noms propres et de sigles.

3.3 Mots et analyse lexicale

Les mots vérifiés par `JEs` sont des chaînes de caractères. Les caractères autorisés forment potentiellement l'ensemble des valeurs du type primitif `char`, c'est à dire l'ensemble des caractères Unicode (près de 39.000 caractères). Les caractères apparaissent dans les arbres lexicaux au fur et à mesure que ceux-ci sont construits. Pour l'anglais, les seuls caractères apparaissant dans les mots sont les lettres de l'alphabet en **minuscule** et **majuscule**, plus le caractère `'` (l'**apostrophe**).

Les mots commençant par une lettre majuscule sont les noms propres ou les sigles (comme `French` ou `Java`), ou bien les mots débutant une phrase, ou encore des mots qui s'écrivent toujours en majuscule (comme `I` et ses dérivés `I've`, `I'd`, etc). Dans tous les cas, `JEs` cherche dans les dictionnaires chargés le mot *avec* la majuscule, et en cas d'échec le mot *sans* la majuscule. Ainsi, dans le texte `God␣save␣the␣Queen` (et dans l'hypothèse où aucun dictionnaire n'est chargé) `JEs` cherche consécutivement les mots `God`, `god`, `save`, `the`, `Queen` et `queen`.

Dans les modes **texte**, **fichier** et **fichier/fichier**, `JEs` doit *extraire* les mots contenus dans des chaînes de caractères qui contiennent des caractères de ponctuation ou même des caractères spéciaux. Cette opération d'extraction s'appelle l'*analyse lexicale*. La seule difficulté concerne ici l'**apostrophe**. L'analyseur lexical doit décider quand ce caractère fait partie d'un mot ou bien quand il faut le considérer comme un séparateur. Ainsi, dans le texte `We'd␣like␣a␣bottle␣of␣'Beaujolais'` `JEs` extrait les mots `We'd`, `like`, `a`, `bottle`, `of` et `Beaujolais`.

Dans le mode **fichier/fichier**, l'extraction des mots à vérifier alterne avec la *recopie* des chaînes de caractères qui séparent ces mots. Par exemple, si le fichier à corriger contient `␣␣My␣God!!...Godzilla␣is␣stil␣alive!!`, `JEs` recopie et vérifie alternativement les chaînes de caractères `␣␣`, `␣`, `!!...`, `␣`, `␣`, `␣` et `!!`, avec les mots `My`, `God`, `Godzilla`, `is`, `stil` et `alive`. La vérification est *aussi* suivie d'une recopie : le mot vérifié est recopié quand il est connu (comme `My`, `God`, `is` et `alive`) ou bien quand il est ignoré (comme `Godzilla`), sinon, un autre mot est copié à sa place (comme `still` qui est copié à la place de `stil`).

4 Architecture

Vous avez libre choix pour organiser votre code et définir les classes que vous jugez utiles. Avant de vous lancer, je vous conseille néanmoins de réfléchir à l'architecture présentée ici et illustrée par la figure 5. Notez que dans l'architectue suggérée, seules certaines classes sont présentées et qu'il est peut-être nécessaire d'en ajouter. De même, seuls certains des attributs de ces classes sont mentionnés.

4.1 La classe `LexicalTree`

Cette classe implémente les arbres lexicaux. C'est dans cette classe qu'on trouve toute l'*algorithmique* du projet. Les méthodes de cette classe sont indépendantes de tout contexte et sont dites de *bas niveau*. Par exemple, la sauvegarde d'un arbre lexical sur disque peut s'effectuer en appliquant la méthode `save(Writer ostream)` à un objet de la classe `LexicalTree`, où `ostream` est un flux pour l'écriture de caractères. Cette méthode n'a aucune idée du nom ni même du type exact de ce flux (`Writer` est une classe abstraite).

4.2 La classe `Dictionary`

Cette classe implémente la notion de dictionnaire. Un dictionnaire est un arbre lexical, plus d'autres informations (comme par exemple le nom du dictionnaire). Les méthodes sont sensiblement les mêmes que celles de la classe `LexicalTree`, mais elles sont de *plus haut niveau*. Par exemple, la sauvegarde d'un dictionnaire sur disque peut s'effectuer en appliquant la méthode `save()` *sans* argument à un objet de la classe `Dictionary` puisqu'on peut calculer le chemin du fichier de sauvegarde à partir du nom du dictionnaire et de la classe `DictionaryFileManager`.

4.3 La classe `DictionaryFileManager`

Cette classe gère la correspondance entre les noms des dictionnaires et les chemins des fichiers correspondants à la sauvegarde de ces dictionnaires sur disque. C'est dans cette classe qu'on définit le chemin du répertoire contenant les fichiers précédemment cités. Notez que le chemin de ce répertoire peut être passé en *paramètre* lors du lancement de `JEs`. Cette classe permet aussi de déterminer le nom du fichier de configuration évoqué en 2.5.

4.4 La classe `SpellChecker`

Une instance de la classe `SpellChecker` est un correcteur complet qui peut vérifier l'orthographe d'*un* mot. Un tel objet possède toutes les fonctionnalités décrite en 2.1.

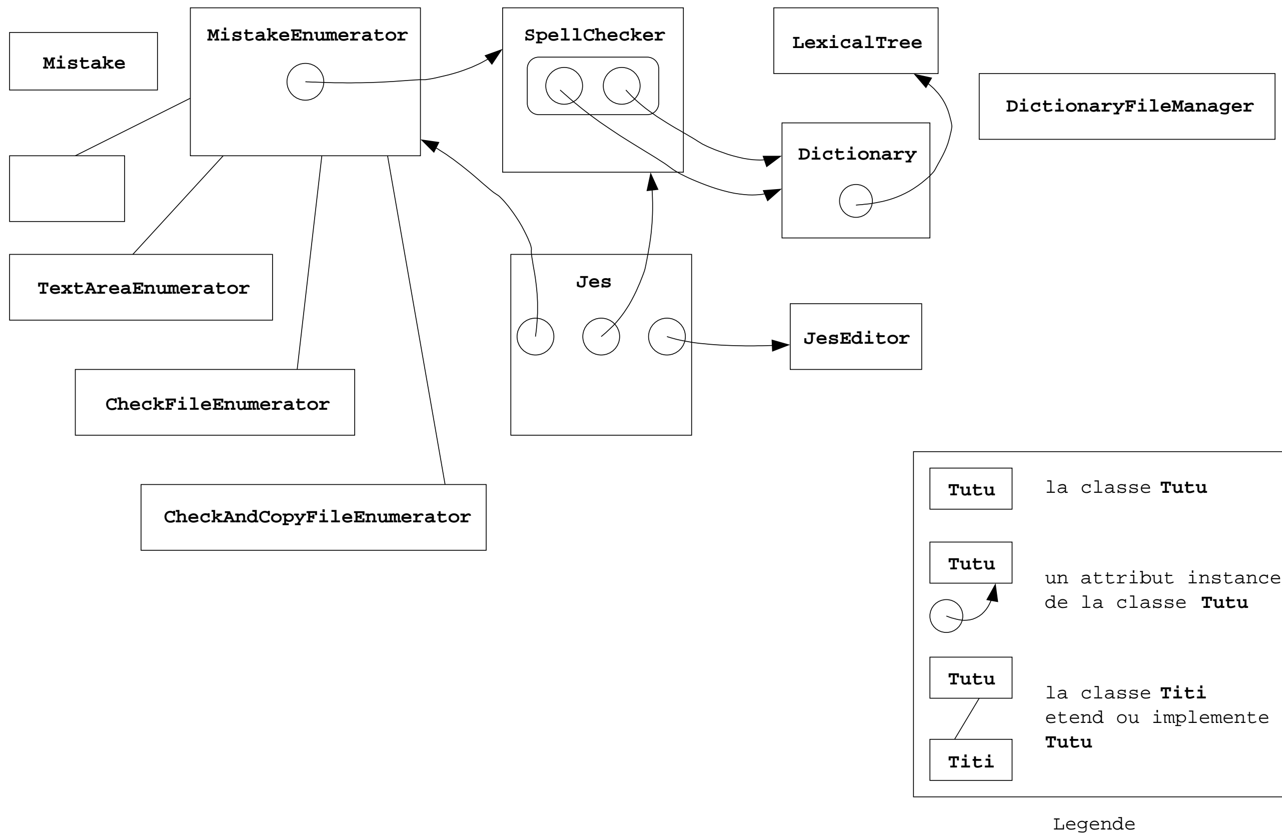


FIG. 5 – *Une architecture possible pour* JEs

4.5 La classe `Mistake`

Un objet de la classe `Mistake` est composé d'un mot (un objet de type `String`), un mot incorrect, et d'un ensemble de mots (par exemple une instance `Vector`), les substitutions possibles pour le mot incorrect. Ce sont des objets de type `Mistake` qui sont produit par la méthode `getMistake()` de la classe `MistakeEnumerator`.

4.6 La classe `MistakeEnumerator`

Cette classe permet d'*abstraire* le comportement des différents *modes* de `JEs`. Que ce soit en mode **texte**, **fichier** ou **fichier/fichier**, on peut voir le comportement du correcteur de manière unifiée : on dispose d'un *objet* qui permet d'*énumérer* les différentes erreurs orthographiques présentes dans (un attribut de) l'objet.

4.7 Les classes `*Enumerator`

Les classes `TextAreaEnumerator`, `FileCheckEnumerator`, `FileCheckAndCopyEnumerator` et *autres classes* de ce type, *étendent* la classe `MistakeEnumerator` et spécialisent ses méthodes en fonction de leur particuliarités. Par exemple, la méthode `replaceBy(String s)` de la classe `TextAreaEnumerator` va remplacer une chaîne de caractères par une autre dans un objet de type `java.awt.TextArea`, alors que cette même méthode pour la classe `FileCheckAndCopyEnumerator` va simplement écrire une chaîne de caractères à la place d'une autre dans un fichier temporaire.

4.8 Les classes `Jes` et `JesEditor`

Les classes `Jes` et `JesEditor` vous sont gracieusement fournies, et vous donnent une interface graphique de base. Si vous utilisez cette interface, vous devez lui ajouter les attributs nécessaires manquants et vous devez compléter les méthodes correspondantes aux différents évènements (clicks bouton ou activations de menu). Dans sa version initiale, l'application est compilable et exécutable et pour chaque évènement, elle imprime simplement un message sur la console.

L'archive `jes.jar` et les fichiers `english.txt` et `english-proper.txt` contenant respectivement les classes fournies et les listes de mots initiales sont accessibles sous le répertoire `/net4/asd101/projet/`

5 Planning

Le tableau ci-dessous propose un agenda possible pour la réalisation du projet, en supposant que vous suivrez l'architecture suggérée dans la partie précédente et que vous utiliserez le squelette d'interface fourni. Durant certaines demi-journées, les plus valeureux de vos chers enseignants passeront dans les salles pour vous encourager et répondre à vos questions. Les demi-journées de passage seront déterminées dynamiquement en fonction de vos besoins, et seront annoncées pendant le projet. Par ailleurs, le *forum* `essi.essi1.jes` sera ouvert pendant toute la durée du projet. Vous pourrez y poster des questions ou des remarques concernant `JEs`, et vos encadreurs tenteront d'y répondre.

	Jeudi	**Vendredi**	**Week-end**	**Lundi**	**Mardi**	**Mercredi**	**Jeudi**	**Vendredi**
Matin	-	étape 2	-	étape 3	étape 4	étape 5	étape 6	étape 7
Après-midi	étape 1	étape 2	-	étape 3	étape 4	étape 5	étape 6	étape 7

- **Etape 1** : vous lisez *attentivement* le sujet et vous testez un correcteur orthographique existant, comme **ispell** ou comme le correcteur de l'éditeur **Word**.
- **Etape 2** : vous implémentez les *arbres lexicaux* et vous les testez avec les listes de mots fournies.
- **Etape 3** : vous implémentez les dictionnaires et leur gestion et vous les testez à l'aide d'une interface textuelle rudimentaire.
- **Etape 4** : vous implémentez les *correcteurs* correspondants au différents modes.
- **Etape 5** : vous finissez les *correcteurs* et vous les testez à l'aide de votre interface textuelle.
- **Etape 6** : vous intégrez votre travail dans l'interface fournie.
- **Etape 7** : vous terminez l'intégration et réalisez les derniers tests. Vous rédigez le fichier `lisezmoi.txt`

6 Remise du projet

6.1 Quoi rendre et comment ?

Vous devez rendre :

- tous les sources `.java` de votre application, sous la forme d'une **unique** archive de nom `jes.jar`. Ne placez dans l'archive que les sources et les fichiers **indispensables** pour votre application (par exemple des ressources). Ne placer dans l'archive **aucun** dictionnaire sous quelque format que ce soit. Commentez votre code pour `javadoc`. Chaque méthode (publique ou privée) devra être totalement documentée (rôle, complexité, pré-conditions, post-relations, invariants). Votre programme devra être compilable et exécutable sans erreur sous *VisualAge* **et** sous le `jdk` (c'est à dire compilable par la commande `javac` et exécutable par la commande `java`) pour JAVA version **1.2**, sous Windows NT et sous Linux. Toutes les classes devront être dans le **seul et unique package** `LOGIN` où `LOGIN` est un des deux logins du binôme.
- un fichier de texte `lisezmoi.txt` dans lequel vous ferez une synthèse de votre travail en indiquant notamment :
  1. les deux noms du binôme
  2. les choses à connaître pour faire marcher votre programme (comme le nom du fichier de configuration)
  3. les particularités que vous avez développées (s'il y en a)
  4. les problèmes que vous avez rencontrés (surtout les plus graves)
  5. vos commentaires sincères sur le projet en général (adéquation avec le cours, niveau de difficulté, etc)

  Ce fichier ne devra pas excéder une page et être au format **texte** (**pas** de `.doc`, `.ps`, `rtf`, etc).

6.2 Quand et où rendre ?

Dans l'**un seulement** de vos deux répertoires `/net4/asd101/renduDeProjet/LOGIN` (où `LOGIN` est le même que celui utilisé pour le nom du package `LOGIN`), vous devez déposer l'archive `jes.jar` et le fichier `lisezmoi.txt` (si ces fichiers sont présents dans les deux répertoires, seul les plus récents seront pris en compte, et vous serez pénalisé pour avoir fait perdre du temps à votre correcteur !).

L'accès au répertoire (et à ses sous-répertoires) `/net4/asd101/renduDeProjet` sera **automatiquement fermé** le

Vendredi 9 Février 2001 à 20h30 heure locale

© *Marc Gaëtano, ESSI 2000-2001*