Introduction :

Ce dernier chapitre est réservé à la phase d'implémentation de la solution maintenue dans le chapitre précédent. Nous exploitons un code OWL extrait de l'ontologie DBpedia sur un programme Java équipé de plusieurs APIs nécessaires comme Jena, WordNet, Lucene. Pour finir, nous testons le nouveau système afin d'évaluer les résultats obtenues.

4.1. Implémentation :

4.1.1. Les outils et les langages utilisés :

- **Java [1]:** est un langage de programmation développé dans les laboratoires de *Sun Microsystems* et fourni gratuitement aux utilisateurs. Nous avons choisis ce langage pour de nombreuses raisons :

✓ Java est un langage orienté objet simple, qui réduit le risque des erreurs d'incohérences.
✓ Il est indépendant de toute plate forme, il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine (JVM).
✓ Il est doté d'une riche bibliothèque de classes, comprenant la gestion des interfaces graphiques (fenêtres, menus, graphismes, boites de dialogue, contrôles)
✓ Il permet d'accéder d'une manière simple aux fichiers et aux réseaux (notamment Internet).
✓ Il est caractérisé aussi par la réutilisation de son code ainsi que la simplicité de sa mise en œuvre.
✓ Il est compatible avec L'API Jena, ce qui nous permet la manipulation, le parcours et la modélisation des documents OWL.

- **Jena [2]:** est un ensemble d'outils (API) open source développés par *HP Labs semantic Web programme,* permettant de lire et de manipuler des ontologies décrites en RDFS ou en OWL et d'y appliquer certains mécanismes d'inférences. Pour notre projet on utilise la version 2.6.4.

- **Lucene [3]:** est une bibliothèque open source écrite en Java qui permet d'indexer et de chercher du texte. Il est utilisé dans certains moteurs de recherche afin de profiter de ces détails techniques sur le traitement du langage naturel (snowball). Le concept est assai

[1] https://www.oracle.com/fr/java/
[2] https://jena.apache.org/
[3] https://lucene.apache.org/

analogue à l'index terminologique que l'on trouve généralement à la fin d'un livre, qui permet de trouver rapidement les pages qui traitent d'un sujet donné.

- **WordNet [4]: est** un thésaurus électronique, une référence lexicale largement utilisée pour l'anglais. Sa conception reposait sur des théories psycholinguistiques des mémoires lexicales humaines. Ce lexique comporte des termes organisés dans des synsets qui regroupent des noms, des verbes, des adjectifs et des adverbes organisés dans des ensembles de synonymes appelés « synsets », représentant les concepts lexicaux sous-jacents.

Pour l'implémentation de mesure de similarité de WordNet on a utilisé l'API JAWS (Java API for WordNet Searching) connectable avec WordNet.

Pour la similarité morphologique, on a utilisé le projet SecondString[5] qui un paquetage java ouvert qui contient plusieurs métriques basées sur la syntaxe comme : JaroWinkler, Levenshtein, QGramsDistance, etc.

4.1.2. L'Implémentation :

a- Edition de l'ontologie DBpedia :

La première étape de l'implémentation consiste à intégrer une copie non complète de l'ontologie DBpedia dans le répertoire principale de l'application , et on a basé sur la présence du concept " Work " et spécialement " Written work " . Nous pouvons accéder au contenu de notre base de connaisses à travers le logiciel (Protégé) ou directement par l'éditeur " Wordpad " afin d'ajouter des propriétés et des instances importantes pour enrichir la base. Il est a signaler que notre DBpedia prototype contient 556 classes, 282 propriétés et 8 instances relatives au concept des auteurs et des livres.

b- Exploitation du code OWL dans le programme Java :

Pour falloir exploiter notre code OWL (fichier DBpedia2014.owl) dans notre programme Java, on s'est rendu compte qu'il existe une API appelée Jena qui sert de lien entre le code OWL et le programme Java. Il faut donc s'assurer de l'implémentation des APIs de la bonne manière et dans le bon emplacement. (Voir Figure 4.1)

[4] https://wordnet.princeton.edu/
[5] http://secondstring.sourceforge.net/

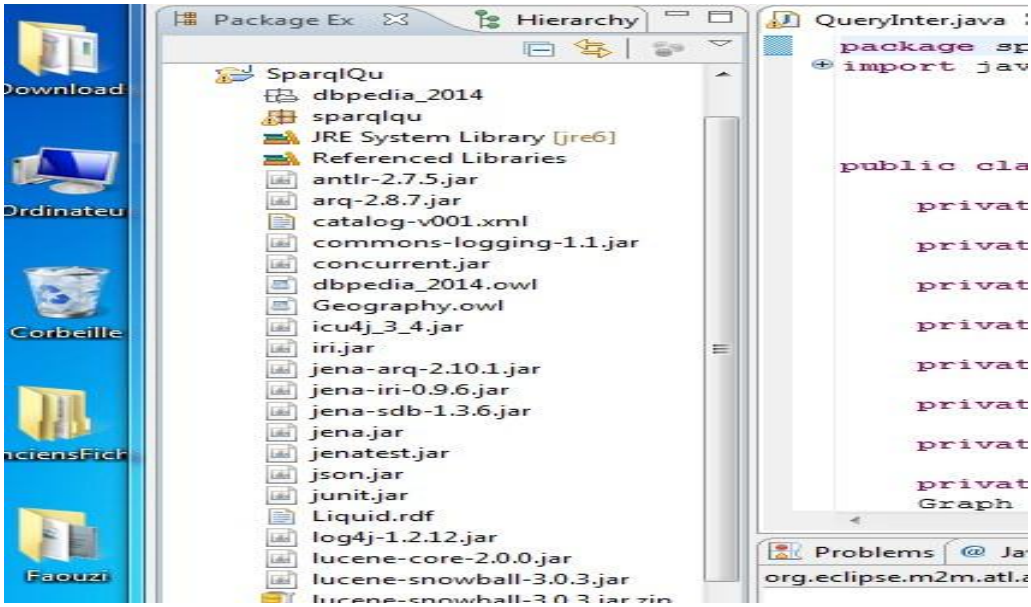


Figure 4.1 : Vue de Librairies utilisés sur JAVA éclipse

c- Exécution de l'application :

Après avoir installer le programme et ces accessoires sur un PC avec un Processeur : Intel Core i5 et Mémoire vive : 4 Go. Après l'exécuté du code , nous avons put capturer les états suivants (voir les figures 4.2 , 4.3 et 4.4) :

- **Interface principale :**

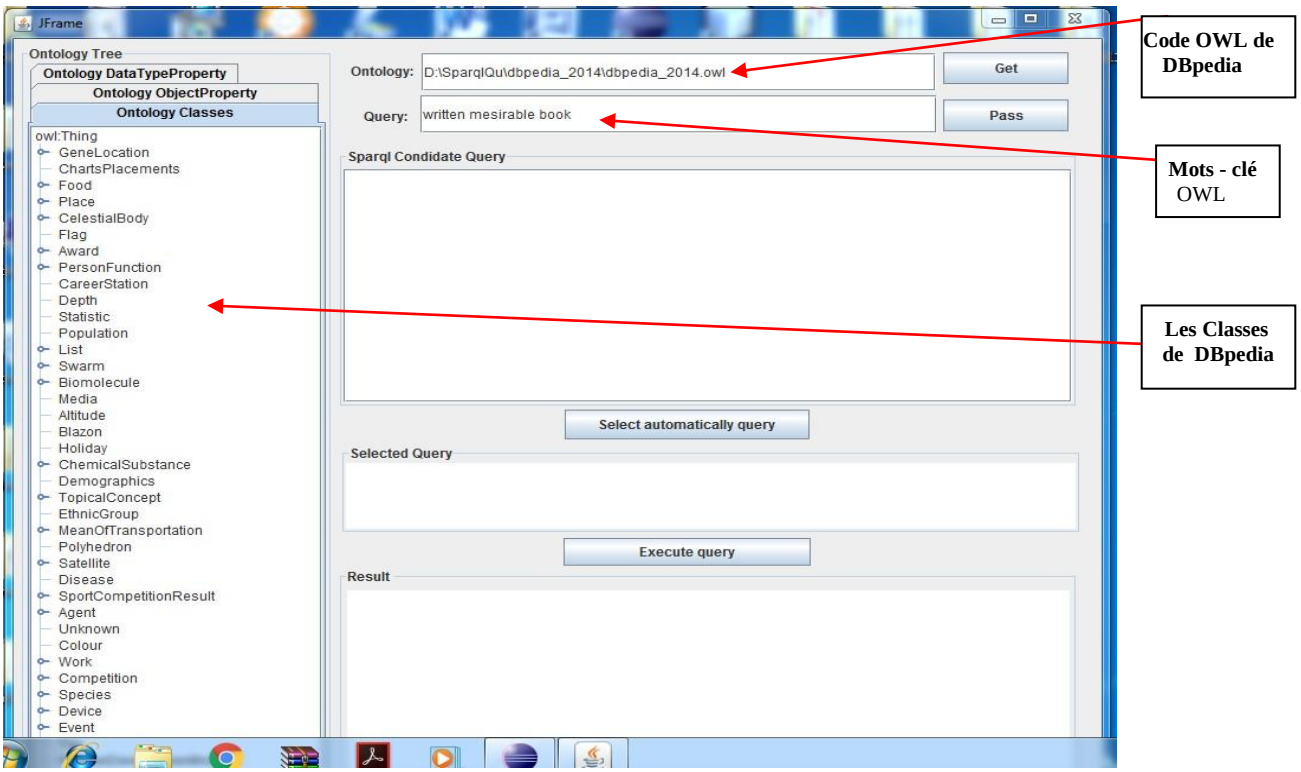


Figure 4.2 : Interface Principal de l'application

- **Interface après la recherche sur le mot –clé " book " :**

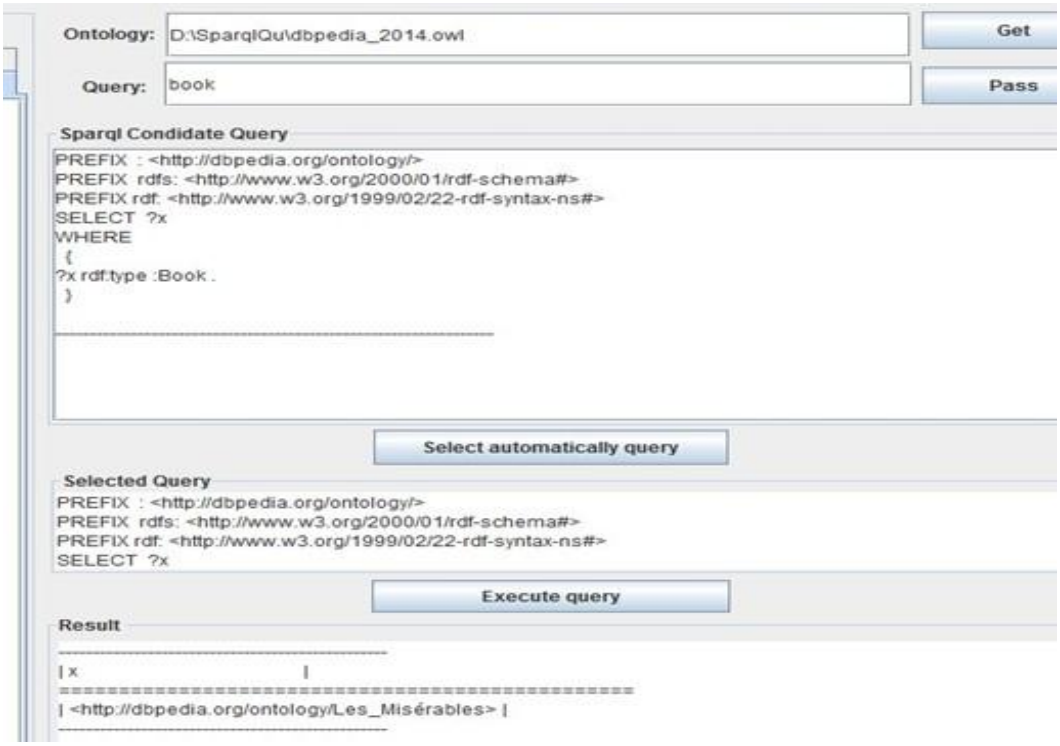


Figure 4.3 : résultat de recherche sur la requête "book"

- **Interface après la recherche sur la requête "** writer misérable book **" :**



Figure 4.4 : résultat de recherche sur la requête " writer misérable book "

4.2. Evaluation :

Pour l'évaluation, nous avons essayé quelques requêtes mots-clés qui sont mentionnées dans le tableau suivant :

N°	*Les Requêtes*
1	*book misérable*
2	*writer book*
3	*writer misérable book*
4	*book*
5	*author misérable book*

Tableau 4.1 : Ensembles des requêtes mots-clés

4.2.1. Métriques utilisées :

Il existe plusieurs facteurs pour mesurer la performance d'un système de recherche d'information tels que: le temps de réponse du système, le nombre de documents pertinent retournés, la méthode de présentation des résultats, ... [BAZ 05].

Mais les deux mesures les plus importantes pour l'évaluation d'un système de recherche est la " Précision " et le " Rappel ". Ces deux mesures sont largement utilisées et permettent de bien connaitre les performances du système de recherche.

a- La précision : permet de déterminer le nombre de Requêtes Pertinentes Retournées par le système de recherche comme résultats par rapport aux nombres Total Requêtes Retournées. La précision parfaite est égale à « 1 », qui veux dire que tous les documents retournés par le système sont pertinents.

$$\textbf{Précision} = \frac{\textbf{Nombre Requêtes Pertinentes Retournées}}{\textbf{Nombre Total Requêtes Retournées}}$$

b- Le rappel : est le Nombre des Requêtes Pertinentes Retournées par le système par rapport aux Nombre Total Requêtes Pertinentes. Le rappel parfait est « 1 », et qui veut dire que tous les documents pertinents sont retournés par le système.

$$\textbf{Rappel} = \frac{\textbf{Nombre Requêtes Pertinentes Retournées}}{\textbf{Nombre Total Requêtes Pertinentes}}$$

Apres l'enregistrement de tous les résultats des recherches faites une par une sur les mots-clés du Tableau 4.1 **,** et on appliquant les formules ci-dessus, nous aurons les résultats suivants (voir tableau 4.2) :

N°	*Requêtes mots-clés*	*Nombre Requête Pertinents Retournés*	*Nombre Total Requêtes Pertinents*	*Nombre Total Requêtes Retournés*	*Rappel*	*Precision*
1	*book misérable*	*1*	*3*	*2*	0.333	0.500
2	*writer book*	*3*	*3*	*4*	1	0.750
3	*writer misérable book*	*2*	*3*	*4*	0.666	0.500
4	*book*	*1*	*3*	*1*	0.333	1
5	*Author misérable book*	*1*	*3*	*2*	0.333	0.500

Tableau 4.2 : Résultats de calcule des Rappels et Précisions

On peut remarquer que notre exemple marque des rappels moyens avec des précisions remarquables.

4.3. Conclusion

Dans ce chapitre, nous avons présenté le détail de l'implémentation de notre application sous un langage Java renforcé avec les API nécessaires pour notre cas. Nous avons préparé la plate forme idéale pour l'exécution du système qui consiste à interroger une copie de la base de connaissances DBpedia par quelques requêtes mots clé choisis. Suite aux résultats obtenus, une évaluation est faite pour juger la pertinence de notre travaille.