Inngéniérie du WEB sémantique
POLYTECH. 5ième année
UNIVERSITÉ D'AIX-MARSEILLE
Année universitaire 2020/2021

Mohamed Ben Ellefi
Pierre Drap
Odile Papini

Feuille de T. P. 4 : Raisonner avec une ontologie en java

Préliminaires

Se connecter sous Linux

S'assurer que la version de java est 1.8 (sinon l'installer)

Télécharger les fichiers compréssés dans AMeTICE :
Section **POUR-TP-ING-WEBSEM**
Celle-ci contient :
- un fichier pizza_1.owl
- un dossier compréssé **library-OWL-API-2020** : qui contient de les librairies utiles pour le TP
- un tutoriel OWL-API

Créer un nouveau projet java (java version 1.8) sous Eclipse
Sauvegarder :
- un tutoriel OWL-API
- toutes les librairies du dossier library-OWL-API -2020
- la documentation OWL-API : `http://owlapi.sourceforge.net/documentation.html` qui contient un tutoriel sor $OWL-API$ et des exemples de code java.
- le tutoriel : `http://owlapi.sourceforge.net/owled2011\_tutorial.pdf`
- exemples de codes : `https://github.com/phillord/owl-api/blob/master/contract/src /test/java/org/coode/owlapi/examples/Examples.java`

ATTENTION : lorsqu'on travaille avec un fichier owl crée avec Protégé sauvegardé dans un répertoire de l'utilisateur, vérifier que l'IRI (sous Linux) du fichier est bien : $file : ///Users/\ chemin\ jusqu'au\ fichier\ /nom\_fichier.owl$

Reprendre la classe java $OWLAPI\_StepByStep$ du TP 3.
Rappel : la classe $OWLAPI\_StepByStep$ contient une classe java $PIZZA$ qui contient les constantes : nom de l'ontologie, IRI, espace de noms.

Rappel : La philosophie de $OWL-API$: une ontologie (ou une OWL ontologie) est une interface de type $OWLOntology$ qui modélise des axiomes ($OWLAxioms$). Une ontologie a un IRI et des méthodes qui lui sont associées. Dans la suite *ontology* désigne une variable de type $OWLOntology$. Lire attentivement les tutoriels avant de commencer à écrire les méthodes suivantes.

— Ecrire une méthode $checkConsistency(OWLOntology\ localOntology)$ qui teste si l'ontologie passée en paramètre est cohérente. Si elle est cohérente elle affiche un message de même si elle est incohérente.

— Ecrire une méthode $validationWithPellet(OWLOntology\ localOntology)$ qui teste si l'ontologie passée en paramètre est incohérente. Si elle est incohérente elle affiche les classes incohérentes. Dans tous les cas, elle affiche l'expressivité en terme de logique de description de l'ontologie

— Ecrire une méthode $generateInferredData(OntoData.ontology, OntoData.manager)$ qui génère l'ontologie inférrée. L'ontologie inférée est stockée dans le fichier $pizza\_1\_D.owl$

— Ecrire une méthode $getInstanceByProperty(OntoData.ontology, OntoData.manager)$ qui permet d'obtenir à partir d'un individu les valeurs des propriétés (Object properties et DataType properties) pour cet individu. Tester avec l'individu $MaRoyaleSansOlive$.

— Ecrire une méthode $printDataFromAnInstance(OntoData.ontology, OntoData.manager)$ qui permet d'obtenir à partir d'un individu, les classes auxquelles appartient cet individu et les valeurs des propriétés pour cet individu. Tester avec l'individu $MaRoyaleSansOlive$.

— Ecrire une méthode $printAllAxiomWithPellet(OntoData.ontology, OntoData.manager)$ qui affiche la hiérarchie de classes (avec indentation) de l'ontologie inférée.

— Ecrire une méthode $printAllIndividualsAndAllTheirProperties(OntoData.ontology)$ qui affiche tous les individus et les valeurs de leurs propriétés.

— Ecrire une fonction *main* pour tester toutes ces méthodes.