Best Of

Actualités les plus lues

Mois
Année

Qu'est-ce que le Web sémantique ?

L'information sur le Web doit être comprise par les ordinateurs, pour que l'on puisse effectuer des requêtes précises sur ces données.
Plus d'informations.

Ressources

- 32 **cours et tutoriels** sur le Web sémantique (dernière mise à jour le **2013-03-13**)
- **FAQ** sur le Web sémantique : 42 QR
- 7 **livres** sur le Web sémantique et 9 **critiques** (dernière mise à jour le **2013-05-30**)

Communauté

- **Le forum** Web sémantique
- **L'équipe** Web sémantique

Facebook

Web sémantique
J'aime 227

Twitter

Suivre @websemantique 90 abonnés

Contacts

Vous souhaitez rejoindre la rédaction ou proposer un tutoriel, une traduction, une question... ? Contactez-nous par MP ou par email (voir en bas de page).

La FAQ sur le Web sémantiqueConsultez toutes les FAQ

Nombre d'auteurs : 8, nombre de questions : 42, dernière mise à jour : 12 septembre 2012

Rechercher

Sommaire→Jena

Précédent Sommaire Suivant

Qu'est-ce que Jena ?

Jena est un ensemble d'outils dédiés à la construction d'applications orientées Web sémantique. Parmi ces outils, on trouve notamment une API Java open-source permettant de manipuler de nombreux langages tels que OWL, RDF/RDFS, SPARQL ou encore N3 et de raisonner sur des modèles ontologiques à l'aide de moteurs d'inférences inclus dans Jena ou externes.

En outre, Jena propose également des systèmes permettant d'assurer la persistance des modèles. On distingue deux systèmes :

- Jena SDB, un magasin de triplets utilisant une base de données relationnelle pour fonctionner ;
- Jena TDB, un système de stockage natif (c'est-à-dire qu'il utilise son propre système de stockage), reconnu comme étant plus performant que son homologue.

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une ontologie OWL ?

Pour stocker en mémoire des ontologies OWL, l'API Jena propose aux utilisateurs la classe *OntModel*. Cette dernière hérite de la classe *Model* permettant de représenter des modèles RDF.

Pour permettre la construction de modèles d'ontologies, Jena met à disposition la classe *ModelFactory*. Le code ci-dessous présente la création d'un modèle d'ontologie sous Jena :

Sélectionnez

```
import com.hp.hpl.jena.ontology.* ;
import com.hp.hpl.jena.ontology.impl.* ;
import com.hp.hpl.jena.rdf.model.* ;

...
OntModel ontologie = ModelFactory.createOntologyModel ();
...
```

Il est possible de paramétrer le sous-langage OWL, le mode de stockage ainsi que le mode d'inférence à employer en utilisant les attributs statiques de la classe *OntModelSpec* en paramètre de la méthode *createOntologyModel ()*. Le tableau suivant présente les possibilités de paramétrage :

OntModelSpec	Langage	Mode de stockage	Raisonneur
OWL_MEM	OWL Full	En mémoire centrale	Aucun
OWL_MEM_TRANS_INF	OWL Full	En mémoire centrale	Par transitivité hiérarchique
OWL_MEM_RULE_INF	OWL Full	En mémoire centrale	Basé sur les règles OWL
OWL_MEM_MICRO_RULE_INF	OWL Full	En mémoire centrale	Basé sur les règles OWL (optimisé)
OWL_MEM_MINI_RULE_INF	OWL Full	En mémoire centrale	Basé sur une partie des règles OWL
OWL_DL_MEM	OWL DL	En mémoire centrale	Aucun
OWL_DL_MEM_RDFS_INF	OWL DL	En mémoire centrale	Basé sur des règles RDFS
OWL_DL_MEM_TRANS_INF	OWL DL	En mémoire centrale	Par transitivité hiérarchique
OWL_DL_MEM_RULE_INF	OWL DL	En mémoire centrale	Basé sur les règles OWL
OWL_LITE_MEM	OWL Lite	En mémoire centrale	Aucun
OWL_LITE_MEM_TRANS_INF	OWL Lite	En mémoire centrale	Par transitivité hiérarchique
OWL_LITE_MEM_RDFS_INF	OWL Lite	En mémoire centrale	basé sur des règles RDFS
OWL_LITE_MEM_RULES_INF	OWL Lite	En mémoire centrale	Basé sur les règles OWL
DAML_MEM	DAML	En mémoire centrale	Aucun
DAML_MEM_TRANS_INF	DAML	En mémoire centrale	Par transitivité hiérarchique
DAML_MEM_RDFS_INF	DAML	En mémoire centrale	Basé sur des règles RDFS
DAML_MEM_RULE_INF	DAML	En mémoire centrale	Basé sur des règles DAML
RDFS_MEM	RDFS	En mémoire centrale	Aucun
RDFS_MEM_TRANS_INF	RDFS	En mémoire centrale	Par transitivité hiérarchique
RDFS_MEM_RDFS_INF	RDFS	En mémoire centrale	Basé sur des règles RDFS

Note : le langage DAML est de plus en plus délaissé au profit du langage OWL notamment.

Par défaut, le modèle créé utilise le langage OWL Full, un stockage en mémoire centrale et un raisonneur basé sur des règles RDFS. Le code ci-dessous présente l'instruction permettant de créer un modèle utilisant le langage OWL DL, un stockage en mémoire centrale et un raisonneur basé sur des règles OWL (par exemple) :

Sélectionnez

```
OntModel ontologie = ModelFactory.createOntologyModel (OWL_DL_MEM_RULE_INF);
```

Le résultat obtenu après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, est le suivant :

Sélectionnez

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

        <owl:Ontology rdf:about="http://www.ontologie.fr/MonOntologie"/>

</rdf:RDF>
```

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une classe OWL ?

Les classes OWL sont représentées par des objets *OntClass* dans l'API Jena. Il nous suffit de procéder de la manière suivante pour créer une nouvelle classe :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création de la classe Voiture
OntClass voiture = ontologie.createClass (namespace + "#Voiture");
```

La fonction *createClass ()* nous permet de créer une classe Voiture dans notre ontologie OWL. Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Voiture"/>
```

Il est à noter qu'il est également possible de créer des classes anonymes en ne précisant pas d'URI en paramètre de la méthode *createClass*.

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une classe union/intersection OWL ?

Les classes unions et intersections sont respectivement représentées par des objets UnionClass et IntersectionClass dans l'API Jena. Il nous suffit de procéder de la manière suivante pour créer une nouvelle classe union :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création des classes
OntClass homme = ontologie.createClass (namespace + "Homme");
OntClass femme = ontologie.createClass (namespace + "Femme");

//Création de la classe union
RDFList listeClasses = ontologie.createList (new RDFNode[]{homme, femme});
UnionClass humain = ontologie.createUnionClass (namespace + "Humain", listeClasses);
```

Ces instructions nous permettent de créer une classe *Humain* union des classes Homme et Femme. Après avoir déclaré ces deux dernières classes, nous créons un objet de type *RDFList* contenant l'ensemble des classes contenues dans l'union. Cette liste est créée à l'aide de la méthode *createList* de l'ontologie. Cette méthode prend en paramètre un tableau de RDFNode, chaque nœud représentant une classe de l'union.

Après avoir créé la liste, il nous suffit d'utiliser la fonction *createUnionClass ()* de l'objet représentant l'ontologie pour créer la classe union. Cette méthode prend en paramètre l'URI de la classe union ainsi que la liste précédemment créée.

Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Femme"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Humain"/>
        <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
                <owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Femme"/>
        <owl:unionOf>
</owl:Class>
```

La création de classes intersections est très similaire à la méthode présentée ci-dessus :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création des classes
OntClass chercheur = ontologie.createClass (namespace + "Chercheur");
OntClass enseignant = ontologie.createClass (namespace + "Enseignant");

//Création de la classe union
RDFList listeClasses = ontologie.createList (new RDFNode[]{chercheur, enseignant});
IntersectionClass enseignantChercheur = ontologie.createIntersectionClass (namespace
```

Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Chercheur"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Enseignant"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#EnseignantChercheur"/>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Chercheur
                <owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Enseignan
        <owl:intersectionOf>
</owl:Class>
```

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une propriété de données OWL ?

Les propriétés de données OWL sont représentées par des objets *DatatypeProperty* dans l'API Jena. L'exemple suivant permet de créer une propriété de données *aPourAge* :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création des classes
OntClass homme = ontologie.createClass (namespace + "Homme");

//Création de la propriété de données
DatatypeProperty aPourAge = ontologie.createDatatypeProperty (namespace + "APourAge"
```

La fonction *createDatatypeProperty ()* nous permet de créer une propriété de données *aPourAge* dans notre ontologie OWL. Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
<owl:DatatypeProperty rdf:about="http://www.ontologie.fr/monOntologie#APourAge"/>
```

Il peut ensuite être intéressant d'enrichir le sens de notre propriété de données en spécifiant un domaine et un objet :

Sélectionnez

```
...
aPourAge.setDomain (homme);
aPourAge.setRange (XSD.xint);
```

La fonction *setDomain ()* est employée pour déclarer que des objets de la classe Homme doivent être utilisés comme sujet de la propriété. La fonction *setRange ()*, quant à elle, permet de spécifier que des données ayant pour type « Entier non signé » doivent être utilisées comme objet de la propriété.

Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:DatatypeProperty rdf:about="http://www.ontologie.fr/monOntologie#APourAge">
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
        <rdfs:domain rdf:resource="http://www.ontologie.fr/monOntologie#Homme"/>
</owl:DatatypeProperty>
```

Il est à noter que dans le cas de la propriété *aPourAge*, un entier a été utilisé comme objet. Il est bien entendu possible de spécifier un autre type de données comme objet de la propriété. Le tableau suivant présente d'autres types pouvant être utilisés comme objet d'une propriété de données :

Ressource Jena	Correspondance RDF
XSD.date	rdf:resource="http://www.w3.org/2001/XMLSchema#date"
XSD.xboolean	rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"
XSD.xbyte	rdf:resource="http://www.w3.org/2001/XMLSchema#byte"
XSD.xdouble	rdf:resource="http://www.w3.org/2001/XMLSchema#double"

XSD.xfloat	rdf:resource="http://www.w3.org/2001/XMLSchema#float"
XSD.xint	rdf:resource="http://www.w3.org/2001/XMLSchema#int"
XSD.xlong	rdf:resource="http://www.w3.org/2001/XMLSchema#long"
XSD.xshort	rdf:resource="http://www.w3.org/2001/XMLSchema#short"
XSD.xstring	rdf:resource="http://www.w3.org/2001/XMLSchema#string"

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une propriété d'objet OWL ?

Les propriétés d'objets OWL sont représentées par des objets *ObjectProperty* dans l'API Jena. L'exemple suivant permet de créer une propriété d'objet *aPourFemme* :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création des classes
OntClass homme = ontologie.createClass (namespace + "Homme");
OntClass femme = ontologie.createClass (namespace + "Femme");

//Création de la propriété d'objet
ObjectProperty aPourFemme = ontologie.createObjectProperty (namespace + "APourFemme"
```

La fonction *createObjectProperty ()* nous permet de créer une propriété d'objet *APourFemme* dans notre ontologie OWL. Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Femme"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
<owl:ObjectProperty rdf:about="http://www.ontologie.fr/monOntologie#APourFemme"/>
```

Il peut ensuite être intéressant d'enrichir le sens de notre propriété d'objet en spécifiant un domaine et un objet :

Sélectionnez

```
...
aPourFemme.setDomain (homme);
aPourFemme.setRange (femme);
```

La fonction *setDomain ()* est utilisée pour déclarer que des objets de la classe *Homme* doivent être utilisés comme sujet de la propriété. La fonction *setRange ()*, quant à elle, permet de spécifier que des objets de la classe *Femme* doivent être utilisés comme objet de la propriété.

Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:ObjectProperty rdf:about="http://www.ontologie.fr/monOntologie#APourFemme">
        <rdfs:range rdf:resource="http://www.ontologie.fr/monOntologie#Femme"/>
        <rdfs:domain rdf:resource="http://www.ontologie.fr/monOntologie#Homme"/>
</owl:ObjectProperty>
```

Créé le 10 juillet 2011 par Yoan Chabot

Comment sérialiser une ontologie OWL ?

Le mécanisme de sérialisation permet d'enregistrer l'ontologie en mémoire dans un fichier. Le code ci-dessous permet de sérialiser le contenu de l'ontologie nommée « ontologie » dans un fichier OWL du même nom :

Sélectionnez

```
...
FileOutputStream fichierSortie = null;
```

```
try {
        fichierSortie = new FileOutputStream (new File ("ontologie.owl"));
}
catch (FileNotFoundException ex) {
        Logger.getLogger (Main.class.getName ()).log (Level.SEVERE, null, ex);
}

ontologie.write (fichierSortie);
```

Le formalisme par défaut utilisé pour la sérialisation est RDF/XML. Il est toutefois possible de choisir un autre formalisme en passant un second paramètre à la méthode *write*. Le tableau suivant présente les valeurs possibles pour ce paramètre ainsi qu'un exemple de résultat obtenu :

Formalisme	Exemple
« RDF/XML »	<rdf:Description rdf:about="http://www.ontologie.fr/monOntologie#Voiture"> <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/> </rdf:Description>
« RDF/XML - ABBREV »	<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Voiture">
« N-TRIPLE »	<http://www.ontologie.fr/monOntologie#Voiture"> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Class> .
« N3 »	<http://www.ontologie.fr/monOntologie#Voiture"> a owl:Class .

Un troisième paramètre de type *String* peut également être utilisé avec la méthode *write*. Celui-ci permet de spécifier une URI utilisée comme base dans l'ontologie. L'utilisation d'une URI de base permet de raccourcir la taille des URI des différentes ressources en utilisant la notation relative plutôt que la notation absolue.

Créé le 13 juillet 2011 par Yoan Chabot

Comment créer une relation hiérarchique entre deux classes OWL ?

La méthode *addSubClass* de la classe *OntClass* est utilisée pour spécifier qu'une classe OWL hérite d'une autre. Le code ci-dessous permet de déclarer que la classe *Voiture* hérite de la classe *Vehicule* :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel ();
String namespace = "http://www.ontologie.fr/monOntologie#";
ontologie.createOntology (namespace);

//Création des classes Voiture et Vehicule
OntClass voiture = ontologie.createClass (namespace + "#Voiture");
OntClass vehicule = ontologie.createClass (namespace + "#Vehicule");

//Déclaration de la relation d'héritage
vehicule.addSubClass (voiture);
```

La fonction *addSubClass ()* nous permet de créer une relation d'héritage entre une classe OWL enfant représentée par l'objet passé en paramètre de la fonction et une classe OWL parent représentée par l'objet qui appelle la fonction. Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
...
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Vehicule"/>
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Voiture"/>
        <rdfs:subClassOf rdf:resource="http://www.ontologie.fr/monOntologie#Vehicule
</owl:Class>
```

Créé le 13 juillet 2011 par Yoan Chabot

Comment charger une ontologie contenue dans un fichier avec Jena ?

Pour réaliser des opérations sur une ontologie contenue dans un fichier OWL, il est nécessaire de charger cette dernière dans un modèle d'ontologie propre à l'API Jena. Le code ci-dessous permet de charger l'ontologie contenue dans le fichier *ontologie.owl* dans un objet de type *OntModel* nommé ontologie :

Sélectionnez

```
&#8230;
FileInputStream in = null;
try
{
   in=new FileInputStream("ontologie.owl");
}
catch (FileNotFoundException ex)
{
   Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
ontologie.read(in,null,null);
```

Le premier paramètre de la méthode read de l'objet de type *OntModel* permet de spécifier la source à utiliser pour charger l'ontologie. Cette source étant un objet de type *InputStream*, il est tout à fait possible de charger une ontologie depuis un flux n'utilisant pas un fichier.

Le second et le troisième paramètre de la méthode read servent respectivement à préciser l'URI de base et le langage à utiliser pour la sérialisation. L'URI de base est utilisée pour convertir d'éventuelles URI relatives de l'ontologie en URI absolues. Si votre ontologie ne contient pas d'URI relatives, ce paramètre peut être de valeur nulle. Le paramètre concernant le langage à utiliser pour réaliser la sérialisation peut également être de valeur nulle dans le cas où le fichier OWL utilise le langage RDF/XML. Dans le cas contraire, il est nécessaire de préciser le langage (« N-TRIPLE » par exemple).

Créé le 25 août 2011 par Yoan Chabot

Comment récuperer l'URI d'une ontologie OWL ?

Le code ci-dessous permet d'afficher les URI des ontologies contenues dans un modèle :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie = ModelFactory.createOntologyModel();
String namespace = &#8220;http://www.ontologie.fr/monOntologie#&#8221;
Iterator iter=ontologie.listOntologies();
while(iter.hasNext())
{
    OntologyImpl onto=(OntologyImpl)iter.next();
    String ontologieURI=(onto.getURI());
    System.out.println(ontologieURI);
}
```

Pour obtenir l'URI de l'ontologie contenue dans le modèle, il nous faut tout d'abord récupérer un objet de type *OntologyImpl* contenant les méta-données relatives à l'ontologie. Pour cela, nous utilisons la méthode *listOntologies()* de notre modèle (nommé ontologie dans l'exemple). Cette méthode nous renvoie un itérateur permettant d'accéder aux différentes ontologies. Dans le cas présent, une seule ontologie est accessible. Toutefois, la méthode proposée ci-dessus permet d'afficher à l'écran les URI de toutes les ontologies contenues dans le modèle. Nous utilisons pour cela une boucle qui s'exécute tant que l'itérateur n'est pas arrivé au bout de la liste des ontologies.

Pour chaque exécution de la boucle, nous récupérons le contenu de l'itérateur dans un objet de type *OntologyImpl* puis nous utilisons la fonction *getURI()* sur ce dernier afin d'obtenir l'URI de l'ontologie actuellement pointée par l'itérateur.

Créé le 25 août 2011 par Yoan Chabot

Comment créer une classe complément OWL avec Jena ?

Les classes compléments OWL sont représentées par des objets *ComplementClass* dans l'API Jena. Il nous suffit de procéder de la manière suivante pour créer une nouvelle classe complément :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie=ModelFactory.createOntologyModel();
String namespace="http://www.ontologie.fr/monOntologie#";
ontologie.createOntology(namespace);
//Création de la classe Homme
OntClass homme=ontologie.createClass(namespace+"Homme");
//Création de la classe complément Femme
ontologie.createComplementClass(namespace+"Femme",homme);
```

La fonction *createComplementClass()* prend en paramètre l'URI de la classe à créer ainsi que la classe complément (celle-ci doit donc être créée au préalable). Le type attendu pour ce second paramètre est *OntClass*. Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abregé, les éléments ajoutés sont les suivants :

Sélectionnez

```
&#8230;
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Femme">
   <owl:complementOf>
      <owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
  </owl:complementOf>
</owl:Class>
```

Créé le 25 août 2011 par Yoan Chabot

Comment créer des instances de classes OWL avec Jena ?

Les instances OWL sont représentées par des objets Individual dans l'API Jena. Il nous suffit de procéder de la manière suivante pour créer une nouvelle instance/un nouvel individu de la classe *Homme* :

Sélectionnez

```
//Création de l'ontologie
OntModel ontologie=ModelFactory.createOntologyModel();
String namespace="http://www.ontologie.fr/monOntologie#";
ontologie.createOntology(namespace);
//Création de la classe Homme
OntClass homme= ontologie.createClass(namespace+"Homme");
//Création d'un homme nommé Paul
homme.createIndividual(namespace+"Paul");
```

La fonction createIndividual() prend en paramètre l'URI de l'instance à créer. Il est également possible de procéder à l'appel de la fonction createIndividual() à partir de notre objet de type OntModel de la manière suivante :

Sélectionnez

```
//Création d'un homme nommé John
ontologie.createIndividual(namespace+"John", homme);
```

Dans le cas ci-dessus, la fonction createIndividual() prend en paramètre l'URI de l'instance à créer et l'URI de la classe à instancier.

Après sérialisation dans un fichier OWL, en utilisant le formalisme RDF/XML abrégé, les éléments ajoutés sont les suivants :

Sélectionnez

```
&#8230;
<owl:Class rdf:about="http://www.ontologie.fr/monOntologie#Homme"/>
<j.0:Homme rdf:about="http://www.ontologie.fr/monOntologie#John"/>
<j.0:Homme rdf:about="http://www.ontologie.fr/monOntologie#Paul"/>
```

Il est à noter qu'il est également possible de créer des instances anonymes en ne précisant pas d'URI en paramètre de la méthode createIndividual().

Créé le 12 septembre 2012 par Yoan Chabot

Précédent Sommaire Suivant