

# TD – Web Sémantique 1

**Julien TSCHERRIG, Joël DUMOULIN, Omar ABOU KHALED**

## **Buts du travail**

- Installation et utilisation de Tomcat 8
- Découverte des outils de base pour le Web Sémantique
- Modélisation de base d'une ontologie
- Création d'une application Java
- Application des notions de bases du sémantique web à l'aide de RDF4J

## **Composants nécessaires**

Composants	Remarques
Java JDK 7	<a href="http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html">http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html</a>
Protégé	<a href="http://protege.stanford.edu">http://protege.stanford.edu</a>
Eclipse JEE	<a href="https://www.eclipse.org/downloads/">https://www.eclipse.org/downloads/</a>
Apache Tomcat 8	<a href="http://tomcat.apache.org/download-80.cgi">http://tomcat.apache.org/download-80.cgi</a>
RDF4J / Sesame	<a href="http://sourceforge.net/projects/sesame/files/Sesame%202/2.7.14/openrdf-sesame-2.7.14-sdk.zip/download">http://sourceforge.net/projects/sesame/files/Sesame%202/2.7.14/openrdf-sesame-2.7.14-sdk.zip/download</a>

## **Travail à réaliser**

- Installation des outils
- Mise en place de Tomcat
- Installation du serveur RDF4J
- Modélisation d'une Ontologie
- Prise en main de RDF4J
- Modélisation d'un graphe à l'aide de RDF4J

# 1 Installation des outils

---

## 1.1 Eclipse

Récupérer la version JEE courante d'Eclipse directement de <https://www.eclipse.org/downloads/>

Décompresser l'archive et lancer « eclipse.exe »

## 1.2 Tomcat

On ne présente plus Tomcat. Il s'agit d'un serveur de Servlet soutenu par Apache. Ce dernier est en perte de vitesse face à son concurrent Glassfish soutenu par Oracle. Contrairement à Tomcat, Glassfish est un serveur d'application, donc plus large et plus lourd que Tomcat. Parmi les applications supportées par Glassfish, on retrouve justement les servlets.

Contrairement à d'autres cours ou nous avons utilisé Tomcat comme endroit de publication (soit pour un Web Service, un site Internet), pour ce TD, nous avons besoin d'une version de Tomcat que nous allons utiliser indépendamment d'Eclipse. Tomcat va être le conteneur pour notre serveur de triplets.

### 1.2.1 Installation de Tomcat

**Attention : Dans tous les cas, veillez à avoir le port 8080 libre**

Récupérer la version 8 de Tomcat à l'adresse suivante : <http://tomcat.apache.org/download-80.cgi>

Prenez la version « Core » en .zip ou directement via le lien ci-dessous.

- <http://apache.spinellicreations.com/tomcat/tomcat-8/v8.0.15/bin/apache-tomcat-8.0.15.zip>

Il suffit d'extraire l'archive pour pouvoir lancer le serveur.

Avant de démarrer le serveur, plusieurs adaptations sont nécessaires :

**Sous Linux ou Mac uniquement**, exécuter ces lignes suivantes : l'objectif est de rendre exécutable les .sh se trouvant sous le /bin du apache-tomcat-8.0.15 décompressé

```
cd apache-tomcat-8.0.15/bin
chmod 777 *.sh
```

**Pour tous les OS**, une petite configuration est nécessaire pour accéder au GUI d'administration. Editer le fichier se trouvant à « apache-tomcat-8.0.15/conf/tomcat-users.xml » et ajouter les lignes suivantes entre les balises « tomcat-users »

```
<role rolename="manager-gui"/>
<user username="admin" password="toto" roles="manager-gui"/>
```

Cela pour but de pouvoir administrer le serveur avec le compte « admin » et mot de passe « toto ».

### 1.2.2 Démarrage de Tomcat

Pour démarrer le serveur :

**Sous Linux ou Mac uniquement**, ouvrir un terminal et rendez-vous à l'emplacement où votre « apache-tomcat-8.0.15 » est décompressé, sélectionner ensuite le répertoire « bin ». Exécuter ensuite la commande suivante :

```
./startup.sh
```

Pour stopper le serveur :

```
./shutdown.sh
```

**Sous Windows uniquement**, rendez-vous à l'emplacement où votre « apache-tomcat-8.0.15 » est décompressé, sélectionner ensuite le répertoire « bin ». Exécuter ensuite « startup.bat »

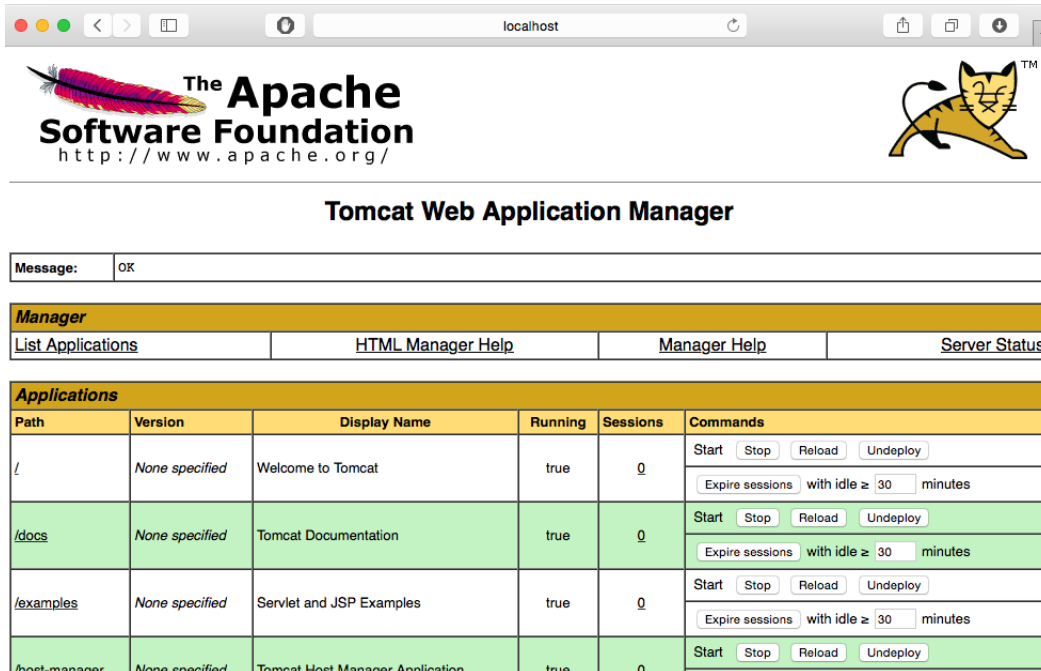
Pour stopper le serveur : shutdown.bat

### 1.2.3 Test de Tomcat

Pour tester le bon fonctionnement de Tomcat, rendez-vous à l'adresse suivante et saisissez les informations utilisateur et mot de passe que nous avons introduites plus haut (admin / toto) :

- <http://localhost:8080/manager/html>

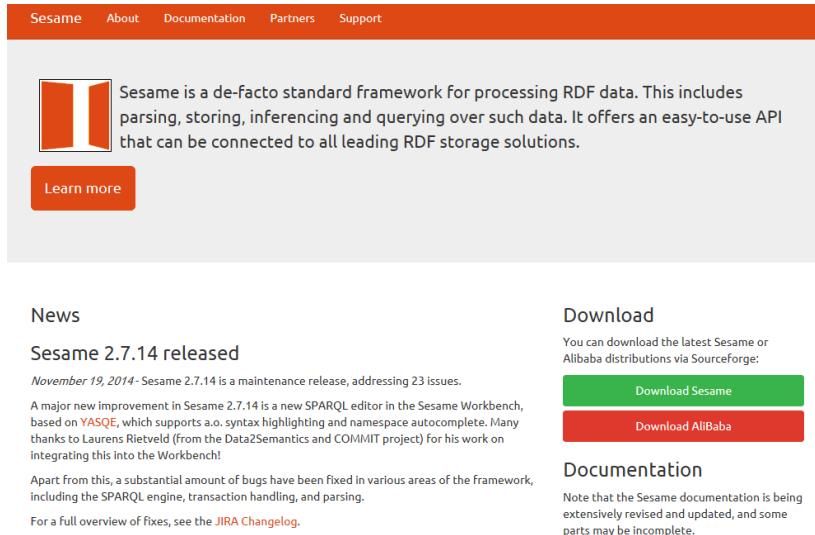
Vous devriez avoir le même écran que ci-dessous. Si c'est le cas, cela signifie que l'installation et la configuration se sont bien passées.



Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle >= 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle >= 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle >= 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy

### 1.3 RDF4J (Sesame)

RDF4J, Sesame ou encore OpenRDF est un Framework pour le traitement de données RDF. Cela comprend l'analyse, le stockage, l'inférence et l'interrogation sur ces données. Il dispose d'une API simple d'utilisation qui peut être connecté à tous les principales solutions de stockage RDF.



The screenshot shows the Sesame website with a navigation bar (Sesame, About, Documentation, Partners, Support). The main content area features a Sesame logo and a description: "Sesame is a de-facto standard framework for processing RDF data. This includes parsing, storing, inferencing and querying over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions." Below this is a "Learn more" button. The "News" section highlights the "Sesame 2.7.14 released" update, dated November 19, 2014, mentioning a new SPARQL editor and bug fixes. The "Download" section provides links to "Download Sesame" and "Download Alibaba". The "Documentation" section notes that the documentation is being extensively revised.

#### 1.3.1 Récupération de RDF4J

Rendez-vous sur le site : <http://rdf4j.org> puis sur la partie du site consacrée au téléchargement.

Pour ce TD, nous allons travailler avec la version 2.7.14. Télécharger ensuite le SDK au format compressé ou utilisé directement le lien ci-dessous :

- <http://sourceforge.net/projects/sesame/files/Sesame%202/2.7.14/openrdf-sesame-2.7.14-sdk.zip/download>

Décompresser ensuite le document compressé.

#### 1.3.2 Déploiement du WAR dans Tomcat

L'installation de RDF4J (serveur), se fait par un déploiement de deux « .war » dans Tomcat soit ceux qui se trouvent dans le dossier « war » du dossier compressé :

- openrdf-sesame.war
- openrdf-workbench.war

Pour déployer un « .war » sur Tomcat, rendez-vous dans le panneau de gestion à l'adresse suivante :

- <http://localhost:8080/manager/html> (vu précédemment)

Dans la section « Deploy » rendez-vous dans la partie « WAR file to deploy ». Sélectionné chacun des deux « .war » ci-dessus à l'aide de « Choose File », puis cliquer sur « Deploy ». Une fois cette manipulation effectuée, vous devriez avoir dans la liste des applications installées.

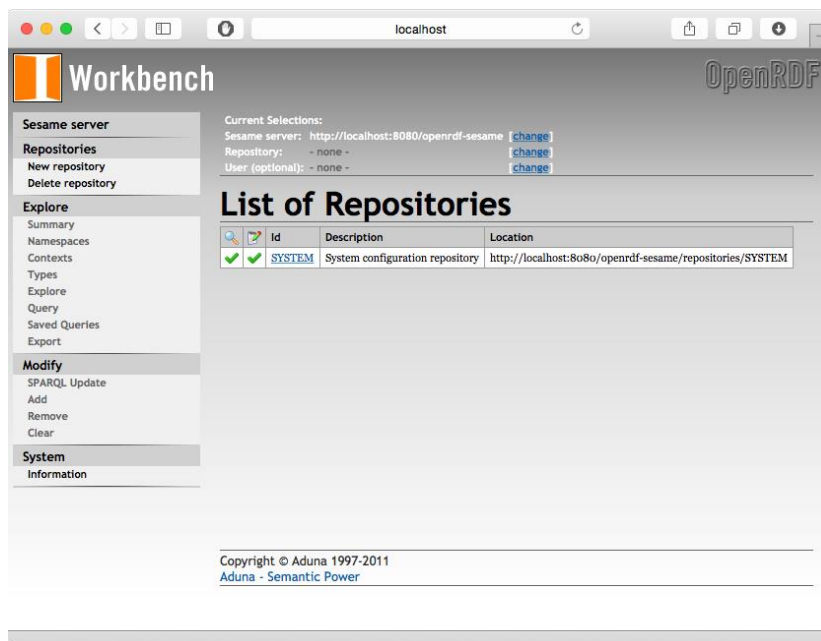
- /openrdf-sesame : serveur RD4J
- /openrdf-workbench : interface de gestion du serveur

### 1.3.3 Test de la partie serveur de RDF4J

Pour tester et s'assurer que RDF4J fonctionne correctement, rendez-vous à l'URL suivante :

- <http://localhost:8080/openrdf%2Dworkbench> - interface graphique pour la gestion du serveur

Vous devriez avoir l'écran suivant :

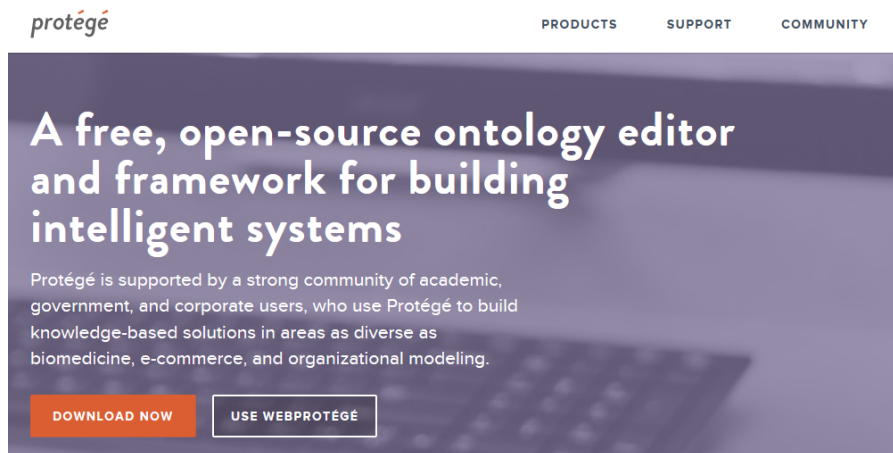


Dans ce cas, l'installation c'est bien déroulée et RDF4J partie serveur est correctement configuré.

## 1.4 Protégé

Protégé Desktop est un environnement riche permettant de créer et éditer une ontologie avec un support complet de l'OWL 2 (langage d'ontologie Web). Il est également possible d'appliquer différents raisonneurs.

Protégé Desktop prend en charge la création et l'édition d'une ou plusieurs ontologies dans un seul espace de travail via une interface utilisateur entièrement personnalisable. Les outils de visualisation permettent de navigation interactive des relations de l'ontologie.



### 1.4.1 Installation de Protégé

Protégé est distribué et maintenu par l'Université de Stanford. Pour télécharger protégé, rendez-vous au lien suivant et téléchargez la version correspondant à votre OS :

- <http://protege.stanford.edu/products.php#desktop-protege>

**Pour Windows, Linux et autre :** L'installation se fait en décompressant l'archive

### 1.4.2 Test de Protégé

**Pour Windows :** lancer protégé en cliquant sur « run.bat » après avoir décompressé l'archive

**Pour Linux et OSX :** lancer protégé à l'aide de la commande suivante :

Il est nécessaire avant la première exécution de donner les droits d'exécution sur le script run.sh de la même manière que nous avons vu plus haut pour Tomcat :

```
chmod 777 run.sh
```

On lance ensuite Protégé à l'aide de la commande suivante :

```
./startup.sh
```

## 2 Modélisation d'une ontologie

### 2.1 Prise en main de protégé

Nous avons vu qu'une ontologie était construite autour de plusieurs éléments construits sous la forme d'arbres :

- **Individu** : Il s'agit d'un élément typé et instancié présent dans l'ontologie.
- **Classes** : Permet de définir un type pour un « Individu »
- **Propriétés Objet (Object Properties)** : la propriété permet de mettre en relation deux « Individus » présents dans l'ontologie à l'aide d'un lien sémantique.
- **Propriétés données (Data Properties)** : la propriété permet de mettre en relation un « Individu » avec une donnée de base (String, Integer, Date, etc.) à l'aide d'un lien sémantique

*Remarque : Tous les éléments présents dans une ontologie sont et doivent être caractérisés par une URI unique !*

C'est précisément les informations que nous retrouvons dans Protégé. Chacun de ces éléments est présent sous forme d'un onglet.

*C'est avec ces 4 onglets uniquement que nous modéliseront une ontologie dans ce TD !*

Pour « Object Properties » et « Data Properties », nous parlons de « Domain » et de « Range ».

- Le « Domain » signifie **à quels types d'éléments (classes)** s'applique cette propriété
- Le « Range » indique **les valeurs possibles** pour cette propriété

### 2.2 Modélisation « HelloWorld »

Vous allez à présent mettre en pratique ce que vous avez vu au cours concernant la création d'ontologie.

#### Exercice :

Modéliser une collection d'articles de presse (par exemple : le 20 minutes<sup>1</sup>)

- Réaliser deux classes : Article et Auteur
- Ajouter les propriétés de type « Data » minimales à Auteur (nom, prénom, etc.)
- Ajouter les propriétés de type « Data » minimales à Article (titre, contenu, date, etc.)
- Ajouter une propriété de type « Objet » indiquant qu'un article a été écrit par un auteur défini
- Créer au moins un article et un auteur et donné leur les propriétés exprimées ci-dessus

*Remarque : N'hésitez pas à être créatif !*

<sup>1</sup> <http://www.20min.ch/ro/>

## 3 Débuter avec RDF4J

Dans cette partie nous allons voir les points suivants :

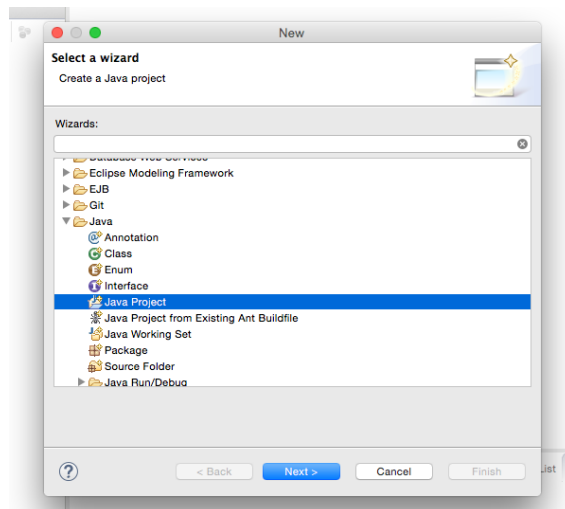
- Création d'un nouveau projet Java dans Eclipse
- Ajout des librairies RDF4J
- Utilisation de RDF4J

### 3.1 Initialisation du projet

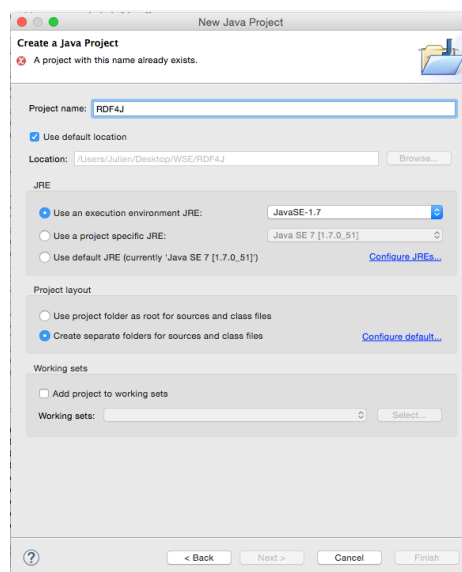
La première étape consiste à créer un projet Java de base.

#### 3.1.1 Création d'un nouveau projet Eclipse

Nous allons débuter par la création d'un nouveau projet Java. Pour cela faites « File » → « New » → « Other ». Sélectionner ensuite « Java Project » sous le répertoire « Java » puis cliquer sur « Next »



Donner ensuite un nom à votre projet (par exemple RDF4J) puis cliquer sur « Finish ».

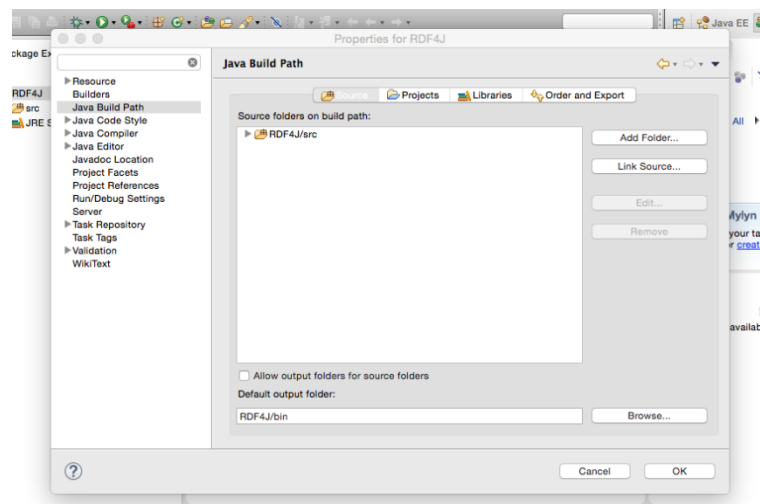




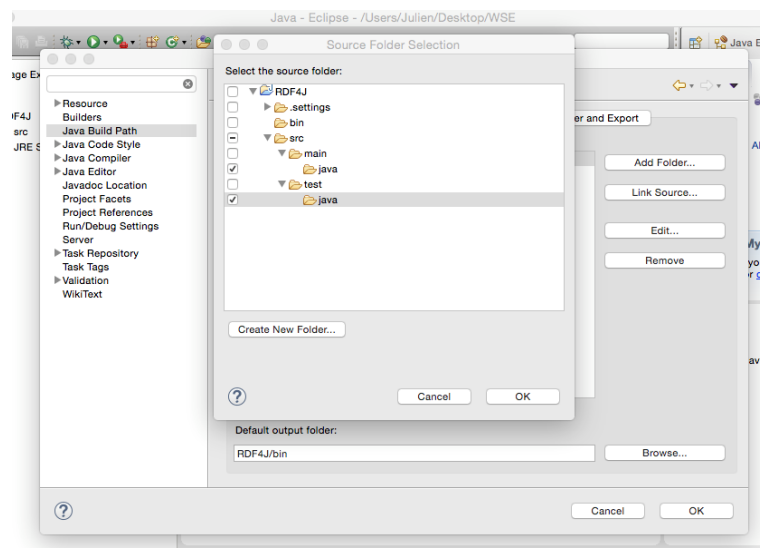
### 3.1.2 Respect des standards JEE

Nous allons à présent créer la structure de base d'une application JEE. Par défaut Eclipse crée une structure de dossier qui lui est propre et qui ne respecte pas les standards du JEE. Seule la ressource « src » existe. Le standard JEE impose d'avoir au minimum une ressource « main/java », celui-ci contiendra le code du programme. Il est possible également de rajouter une ressource « test/java », celui-ci contiendra les classes de tests du programme.

Pour arriver à ce résultat Faites un clic droit sur le projet « RDF4J » → « Properties » → « Java Build Path » puis sélectionner l'onglets « Source ». Cliquer ensuite sur le bouton « Add Folder... »

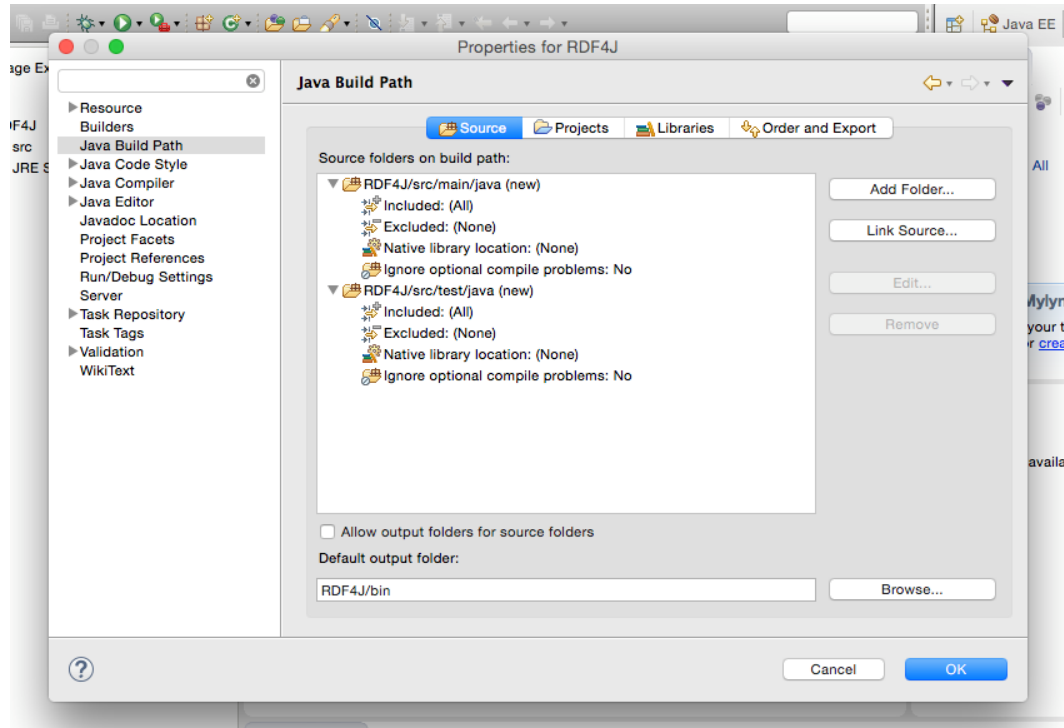


Sélectionner ensuite le répertoire « src » déjà existant dans la structure, puis cliquer sur « Create New Folder... ». Ajouter le répertoire « main/java » puis cliquer sur « Finish ». Répéter ensuite cette dernière opération pour y ajouter le répertoire « test/java ». Vous devriez arriver au résultat suivant :



Fermer la fenêtre à l'aide du bouton « OK ». Une erreur vous est alors signalée. Sélectionner dans l'onglet source « RDF4J/src » puis cliquer sur le bouton « Remove ». L'erreur n'est plus présente.

Vous devriez obtenir le résultat suivant :



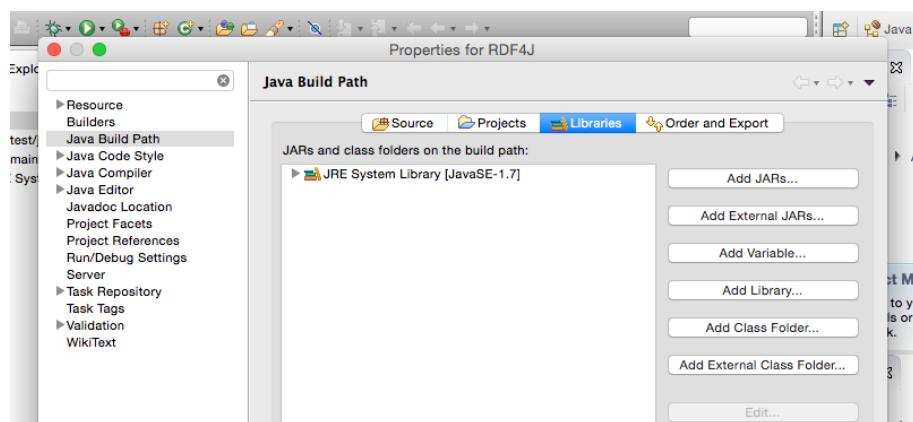
Vous avez à présent la structure correcte d'un projet JEE. Pour terminer et valider vos modifications cliquer sur le bouton « OK ».

### 3.1.3 Ajout des libraires

Au point 1.3.1 vous avez téléchargé les librairies nécessaires à son utilisation. Le SDK téléchargé contient de nombreux fichiers mais seuls les éléments se trouvant dans le répertoire « lib » sont ici nécessaires.

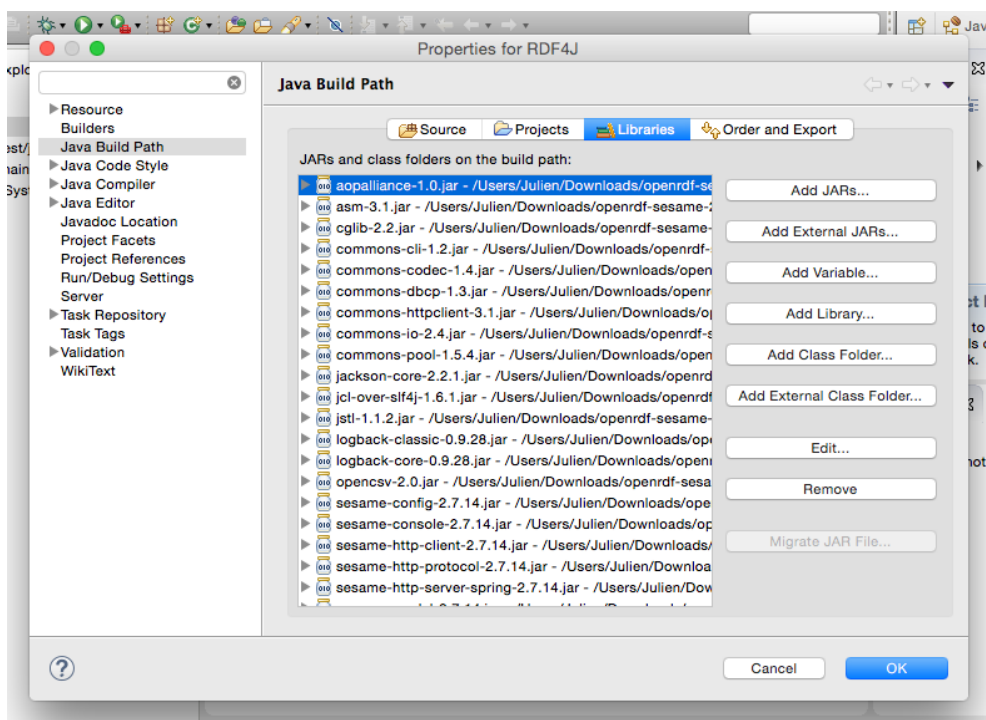
Name	Date Modified	Size	Kind
▶ docs	19 Nov 2014 16:23	62.2 MB	Folder
▶ war	19 Nov 2014 16:23	10.5 MB	Folder
▶ lib	19 Nov 2014 16:23	7.3 MB	Folder
NOTICE.txt	9 Oct 2014 17:11	2 KB	text
LICENSE.txt	21 Jan 2014 12:00	2 KB	text
▶ bin	19 Nov 2014 16:23	1 KB	Folder

Il est nécessaire maintenant d'ajouter toutes ces libraires au projet. Pour cela, fait un clic droit sur le projet « RDF4J » → « Properties » → Java Build Path » et sélectionner l'onglet « Librairies ».



Cliquer ensuite sur le bouton « Add External JARs... » et rendez-vous dans le dossier de RDF4J téléchargé (openrdf-sesame-2.7.14) puis entrer dans le répertoire « lib ». Sélectionner tous les « .jar » contenu dans ce répertoire et cliquer sur « Open ».

Vous devriez arriver au résultat suivant :



Vous avez à présent l'ajout des libraires en cliquant sur le bouton « OK ».

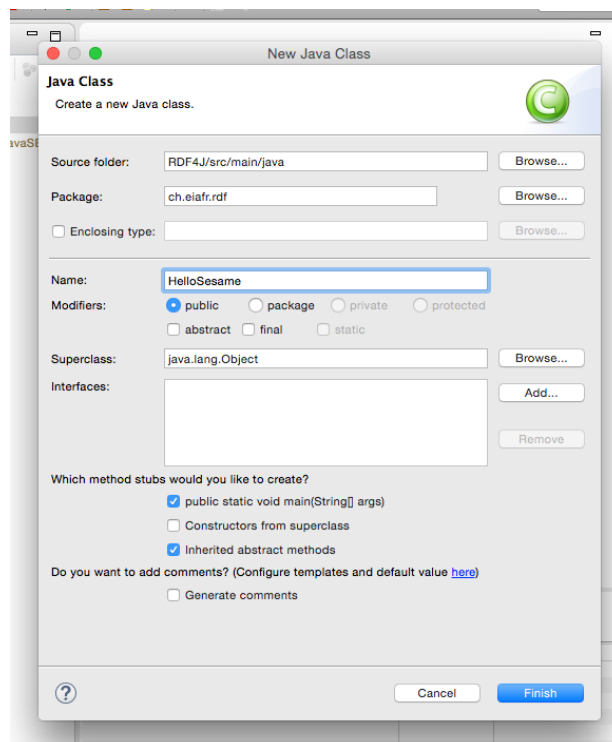
## 3.2 Manipulation des données avec RDF4J

Nous allons à présent voir comment manipuler des informations au format RDF(S) avec RDF4J.

### 3.2.1 Création de la classe "HelloSesame"

Nous avons terminé la préparation, nous pouvons à présent commencer la programmation. Faites un clic droit sur « src/main/java » et sélectionnez « New » -> « Class ». Dans la boîte de dialogue affichée, définir le nom du package à ch.eiafr.rdf, le nom de classe à HelloSesame

*Remarque : Vérifiez que l'option public static void main (String [] args) est cochée.*



Cliquez sur «Terminer» et Eclipse va créer la nouvelle classe et l'ouvrir dans un éditeur automatiquement.

### 3.2.2 Ajout et initialisation du MemoryStore

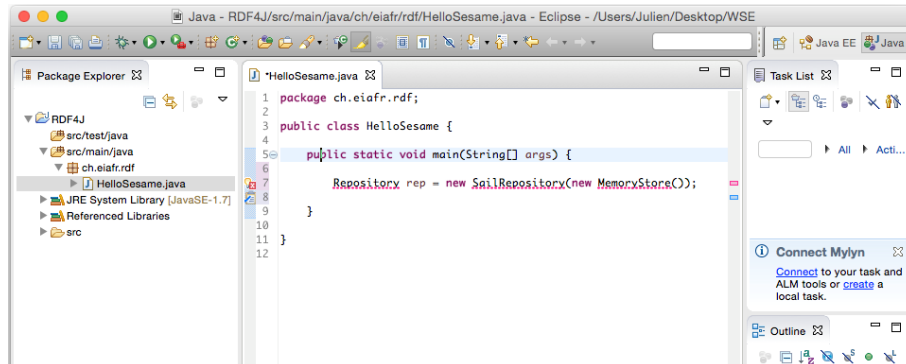
Puisque nous voulons travailler avec des données RDF, nous allons commencer par créer un store RDF. Il s'agit d'une base de données triple, spécialement conçue pour contenir les données RDF.

*Remarque : Dans sésame, les données RDF sont généralement stockées dans un dépôt type « triple store ». Il existe de nombreux types de dépôt (accès local, accès distant, avec ou sans persistance de données, etc.).*

Dans ce tutoriel, nous allons créer un dépôt local simple, soit celui qui utilise un store en mémoire, sans aucune sorte de fonctionnalité de stockage persistant (les données seront perdues lors de la sortie du programme). Ajoutez le code suivant à votre main :

```
Repository rep = new SailRepository(new MemoryStore());
```

Une fois que vous avez fait cela, vous remarquerez quelques lignes rouges apparaissent



Cela signifie que quelque chose ne va pas avec votre code. Dans ce cas, le problème est que plusieurs déclarations d'importation sont absentes. Il est nécessaire d'ajouter celles-ci. Vous pouvez ajouter manuellement chaque importation, mais heureusement Eclipse vous aide à résoudre certains problèmes – en particulier les problèmes d'import. Faites un clic sur l'ampoule à gauche de la ligne. Une boîte de dialogue apparaît avec les possibilités. Sélectionnez ensuite la première possibilité pour résoudre le problème.

Maintenant que nous avons créé notre référentiel, nous avons besoin de l'initialiser. Ceci est fait en appelant la méthode suivante :

```
rep.initialize();
```

Vous devriez arriver à obtenir ainsi les dépendances suivantes :

```
import org.openrdf.repository.Repository;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.memory.MemoryStore;
```

### 3.2.3 Ajout de données dans le dépôt

Maintenant que nous avons créé et initialisé notre dépôt, nous allons ajouter des données dans celui-ci. L'ajout de données peut se faire de plusieurs façons. Dans ce TD, nous allons créer quelques déclarations RDF directement en Java (il est également possible de les charger à partir d'un fichier).

Pour ce faire, nous avons besoin de quelques informations complémentaires

- Un espace de noms - utilisé pour créer de nouveaux URI dont nous avons besoin
- Une valueFactory - utilisé pour créer des URI,
- Des « Binary node » - éléments du dépôt
- Des « objets littéraux » - éléments du dépôt

A l'aide des lignes suivantes, commençons tout d'abord par indiquer notre espace de nom « namespace » ainsi que récupérer notre « ValueFactory »

```
String namespace = "http://eia-fr.ch/";
ValueFactory f = rep.getValueFactory();
```

Avec ces éléments, nous pouvons créer une nouvelle URI. Nous allons créer un identificateur (URI) pour une personne appelée « John » (individu), dont nous allons ajouter plus tard des données à notre dépôt.

```
URI john = f.createURI(namespace, "julien");
```

Nous avons créé une URI, mais n'avons pas encore ajouté des données à notre dépôt. Pour ce faire, nous devons d'abord ouvrir un « RepositoryConnection ». Cette connexion nous permet d'ajouter, de supprimer et récupérer des données à partir de notre dépôt. Nous faisons cela comme suit :

```
RepositoryConnection conn = rep.getConnection();
```

Il est important de conserver notre connexion ouverte et de la fermer à nouveau lorsque nous aurons fini avec eux pour libérer la ressource. A cette fin, nous pouvons créer un bloque try-finally où tout ce que nous voulons faire avec la connexion qui se passe dans la clause « try », et la connexion est fermée dans la clause « finally » (de cette manière, s'il devait y avoir une erreur, la connexion serait automatiquement fermée).

**Attention : vérifié que vos « imports » utilisent les bons « packages »**

Votre code devrait maintenant se présenter de cette manière :

```
11 public class HelloSesame {
12
13     public static void main(String[] args) throws RepositoryException {
14
15         Repository rep = new SailRepository(new MemoryStore());
16         rep.initialize();
17
18         String namespace = "http://eia-fr.ch/";
19
20         ValueFactory f = rep.getValueFactory();
21
22         URI john = f.createURI(namespace, "julien");
23
24         RepositoryConnection conn = rep.getConnection();
25
26         try {
27             // TODO put your code here !
28
29         } finally {
30             conn.close();
31         }
32     }
33 }
34
35 }
```

Grâce à la connexion, nous pouvons commencer à ajouter des déclarations à notre dépôt. Nous allons ajouter deux triples :

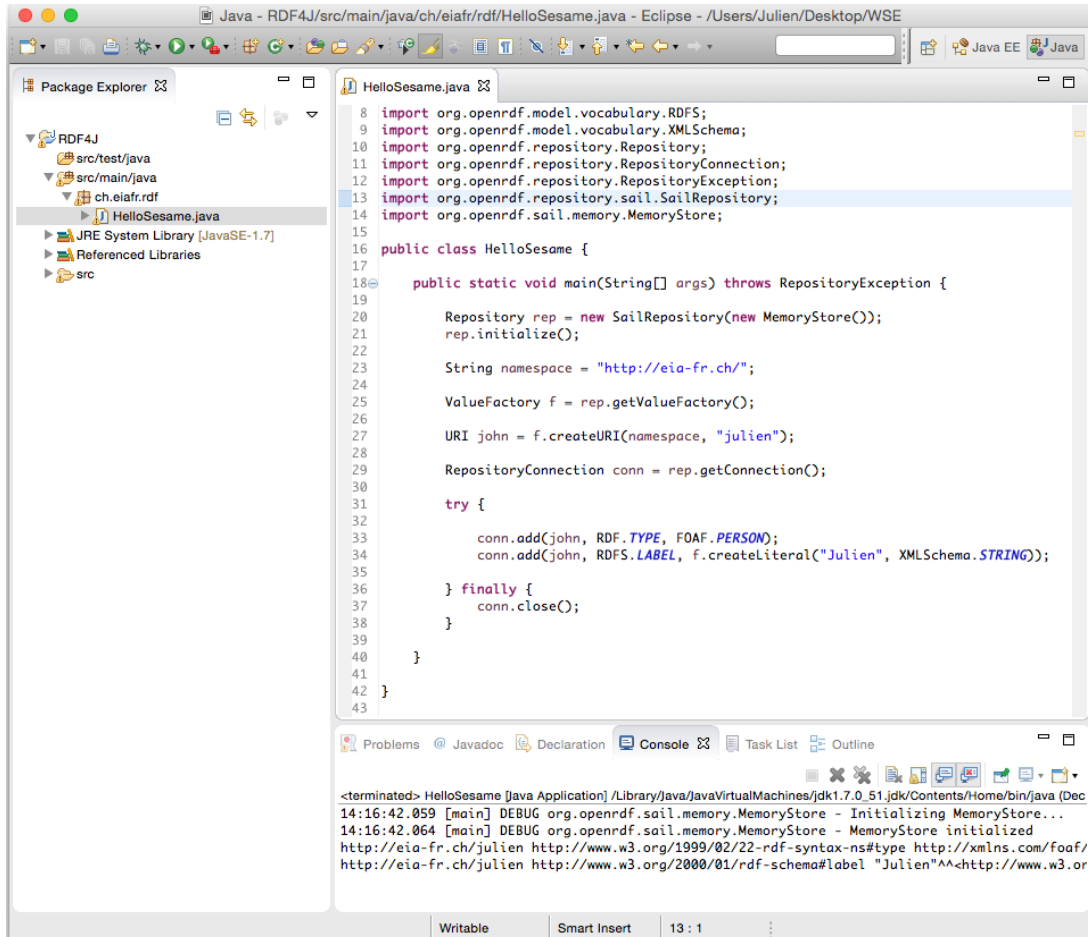
- Le premier informe que l'individu John est du type « PERSON » décrit dans FOAF<sup>2</sup>
- Le second informe que le nom de l'individu John (URI) est « John »

```
conn.add(john, RDF.TYPE, FOAF.PERSON);
conn.add(john, RDFS.LABEL, f.createLiteral("John", XMLSchema.STRING));
```

<sup>2</sup> <http://www.foaf-project.org>

Noter comment pour vocabulaires RDF connus (RDF, RDFS et FOAF), Sesame fournit des constantes que vous pouvez facilement réutiliser pour créer / lire les données avec. En outre, vous voyez une autre utilisation de la « ValueFactory » ici, cette fois pour créer un objet littéral.

Votre code devrait maintenant se présenter de cette manière :



```
8 import org.openrdf.model.vocabulary.RDFS;
9 import org.openrdf.model.vocabulary.XMLSchema;
10 import org.openrdf.repository.Repository;
11 import org.openrdf.repository.RepositoryConnection;
12 import org.openrdf.repository.RepositoryException;
13 import org.openrdf.repository.sail.SailRepository;
14 import org.openrdf.sail.memory.MemoryStore;
15
16 public class HelloSesame {
17
18     public static void main(String[] args) throws RepositoryException {
19
20         Repository rep = new SailRepository(new MemoryStore());
21         rep.initialize();
22
23         String namespace = "http://eia-fr.ch/";
24
25         ValueFactory f = rep.getValueFactory();
26
27         URI john = f.createURI(namespace, "julien");
28
29         RepositoryConnection conn = rep.getConnection();
30
31         try {
32             conn.add(john, RDF.TYPE, FOAF.PERSON);
33             conn.add(john, RDFS.LABEL, f.createLiteral("Julien", XMLSchema.STRING));
34         } finally {
35             conn.close();
36         }
37     }
38 }
39
40 }
41
42 }
43 }
```

<terminated> HelloSesame [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0\_51.jdk/Contents/Home/bin/java (Dec 14:16:42.059 [main] DEBUG org.openrdf.sail.memory.MemoryStore - Initializing MemoryStore... 14:16:42.064 [main] DEBUG org.openrdf.sail.memory.MemoryStore - MemoryStore initialized http://eia-fr.ch/julien http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://xmlns.com/foaf/ http://eia-fr.ch/julien http://www.w3.org/2000/01/rdf-schema#label "Julien"^^http://www.w3.or

Vous savez à présent comment ajouter des triples dans RDF4J.

### Exercice :

A l'aide des connaissances acquises pour l'ajout, modéliser un individu contenant vos informations en vous basant avec le vocabulaire FOAF<sup>3</sup>.

Utiliser uniquement les propriétés suivantes : name, nick, phone, homepage

### Question :

- A quoi sert le label (RDFS.LABEL) utilisé plus haut ?
- Quel est la différence entre RDF et RDFS ?

<sup>3</sup> <http://xmlns.com/foaf/spec/>

### 3.2.4 Récupération de données présentes dans le dépôt

Une fois les données ajoutées au dépôt, nous pouvons les récupérer de retour à partir de celui-ci. Vous pouvez le faire de plusieurs façons. Dans ce TP nous utiliserons directement les méthodes de l'API de RepositoryConnection.

*Remarque : Une autre méthode consisterait à utiliser le langage SPARQL. Cette méthode sera vue la semaine prochaine.*

Pour réaliser cette opération nous allons utiliser la méthode « `getStatements()` » de la façon suivante.

```
RepositoryResult<Statement> comptes = conn.getStatements (null, null, null, true);
```

Nous utilisons la méthode « `getStatements()` » pour récupérer des données à partir du dépôt. Les trois premiers arguments (tous nuls). Ils représentent le sujet, le prédicat et l'objet que nous souhaitons faire correspondre. Comme ils sont tous mis à null, cela va récupérer tous les états.

*Remarque : Le dernier argument est un booléen. Il a pour but pour indiquer si les déclarations présumées par un raisonneur devraient être inclus. Puisque nous ne avons pas raisonneur configuré sur notre boutique, modifier ce paramètre ne aura aucune influence sur le résultat. Plus d'informations sur le raisonnement vous seront communiqués la semaine prochaine.*

L'objet retourné par « `getStatements()` » est un « `RepositoryResult` », qui est un type d'itération. Celui-ci vous permet de traiter le résultat dans un mode streaming, utilisant la méthode « `hasNext()` ».

Remarque : Presque toutes les méthodes qui extraient des données à partir du dépôt renvoient un « `Iterator` ». Le travail avec des « `Triple Store` » peut retourner très rapidement un grand nombre de résultats. Ceux-ci ne peuvent le plus souvent pas être enregistré en mémoire en même temps (vitesse d'exécution et espace mémoire).

#### Exercice :

A l'aide de l'itérateur, afficher les informations (sujet, prédicat et objet) pour chaque « `Statement` » de la méthode « `getStatements` » proposée ci-dessus.

#### Questions :

- Que pouvez-vous dire du sujet affiché ?
- Que pouvez-vous dire du type d'affichage de l'objet ?

### 3.2.5 Ajout d'un modèle

Notre dépôt ne contient que deux déclarations, nous pouvons en toute sécurité simplement transférer la totalité résultat dans une collection Java. RDF4J possède une interface « `Model` » pour traiter les données RDF. Il est possible de convertir un résultat en « `Model` » à l'aide de la méthode suivante.

```
Model model = Iterations.addAll(statements, new LinkedHashModel());
```

Cette méthode récupère toutes les déclarations de l'itération fourni, les met dans un modèle donné (dans notre cas : `LinkedHashModel`) et retourne le « `Model` ».



Il est possible d'afficher les données contenues dans ce « Model » à l'aide de la méthode suivante.

```
Rio.write(model, System.out, RDFFormat.TURTLE);
```

Rio est une classe propre à RJF4J du type analyser / writer. Nous pouvons simplement passer notre modèle, spécifiez le flux sortant, et le format dans lequel nous aimerions qu'il a envoyée, et Rio fait le reste. Votre code devrait maintenant ressembler à ceci (dans notre cas, il s'agit d'un format Turtle, n3 ou encore notation 3)

```
RepositoryResult<Statement> statements = conn.getStatements (null, null, null, true);  
Model model = Iterations.addAll(statements, new LinkedHashModel());  
Rio.write(model, System.out, RDFFormat.TURTLE);
```

Exécuter en tant -> application Java. Vous verrez la sortie suivante dans la console. Vous venez à présent de créer une première application basée sur le web sémantique.

```
<http://eia-fr.ch/julien> a <http://xmlns.com/foaf/0.1/Person> ;  
    <http://www.w3.org/2000/01/rdf-schema#label>  
        "Julien"^^<http://www.w3.org/2001/XMLSchema#string> .
```

### 3.2.6 Ajout de namespaces

Cependant, le code n'est pas très lisible en utilisant à chaque fois des url complètes.. Nous pouvons simplifier la sortie, en ajoutant quelques définitions d'espace de noms à notre modèle à l'aide des lignes suivantes :

```
model.setNamespace("rdf", RDF.NAMESPACE);  
model.setNamespace("rdfs", RDFS.NAMESPACE);  
model.setNamespace("xsd", XMLSchema.NAMESPACE);  
model.setNamespace("foaf", FOAF.NAMESPACE);
```

Ajoutez ces lignes à votre code, après où vous avez créé le modèle. Ensuite, exécutez à nouveau votre programme. Vous verrez alors la suivante:

```
http://eia-fr.ch/julien a foaf:Person ;  
    rdfs:label "Julien"^^xsd:string .
```

#### Exercice :

Ajouter un namespace pour « <http://eia-fr.ch/> » et tester la sortie. Quel est la différence ?

### 3.3 RDF4J avec persistance

Nous avons vu jusqu'à présent l'utilisation de la librairie RDF4J utilisant un triple store (dépôt) local et non persistant.

```
Repository rep = new SailRepository(new MemoryStore());  
rep.initialize();
```

Cette utilisation n'est pas conseillée lors de réalisation de projet. Dans ce dernier point, vous allez utiliser un triple store (dépôt) local et persistant ainsi qu'un triple store distant (installé sur un serveur Tomcat).

#### 3.3.1 Local persistant

Le fonctionnement d'un « Repository » local persistant se base sur un « MemoryStore » standard. Il suffit d'indiquer où la base de donnée sera sauvegardée

```
File dataDir = new File("C:\\temp\\myRepository\\");  
Repository rep = new SailRepository(new MemoryStore(dataDir));  
rep.initialize();  
  
...  
  
Rep.close() ;
```

#### Exercice :

Essayer la persistance sur un « MemoryStore » locale

Que se passe-t-il lorsque l'on souhaite ajouter plusieurs fois le même « triple » (sujet, prédicat, objet) ?

#### 3.3.2 Triple store distant

Nous avons au point 1.2 procédé à l'installation de Tomcat et au point 0 déployer un serveur sémantique RDF4J sur notre serveur Tomcat. L'objectif est maintenant de faire communiquer en enregistrant et récupérant les résultats directement depuis le serveur.

Comme on a pu le voir dans le point 3.3.1, nous n'avons eu qu'à modifier le « Repository » pour que tout le système soit persistant. Il en va de même pour indiquer au système que le store sera distant.

#### Exercice :

A l'aide de la documentation mise à disposition par RDF4J disponible via son site Internet<sup>4</sup>, mettez en place la connexion distante.

- Créer un « Repository » sur le serveur (via l'interface d'openrdf-workbench)
- Le string d'accès au serveur est : <http://localhost:8080/openrdf-sesame> (car installé sur localhost)
- Aucun credential n'est nécessaire par défaut

---

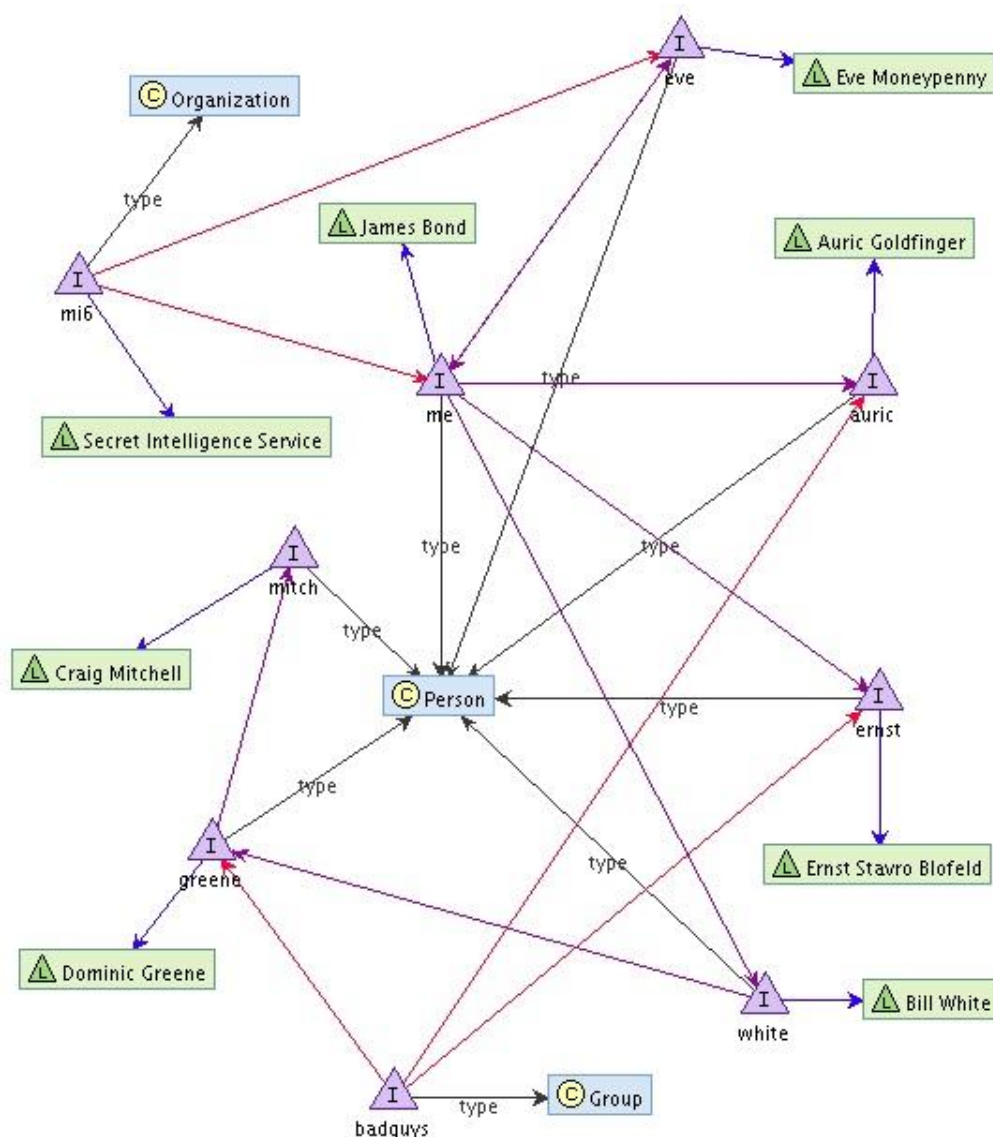
<sup>4</sup> <http://rdf4j.org/sesame/2.7/docs/users.docbook?view>

### 3.4 Modélisation d'un graphe

#### Exercice :

Vous disposez ci-dessous d'un graphe représentant différentes entités présentes dans la saga « James Bond ». Représentez-le à l'aide du vocabulaire FOAF<sup>5</sup>.

- Les arcs violets représentent la relation « knows » présente dans FOAF
- Les arcs rouges représentent la relation « member » présente dans FOAF
- Les arcs bleus représentent la relation « name » présente dans FOAF



<sup>5</sup> <http://xmlns.com/foaf/spec/>

### 3.5 Conclusion

Vous avez à présent vu la manipulation de données RDF(S) grâce à la librairie RDF4j - aussi connue sous le nom de Sesame ou encore OpenRDF.

Vous savez maintenant :

- Initialiser un Triple Store
- Ajouter de nouveau individu
- Récupérer les individus et les afficher
- Manipuler un modèle

## 4 Pour aller plus loin

---

En vous basant sur le graphe que vous avez réalisé au point 3.4, essayé à l'aide de la méthode « `getStatements()` » vue plus haut de répondre aux questions suivantes :

```
RepositoryResult<Statement> comptes = conn.getStatements (null, null, null, true);
```

- Quelles sont les noms de toutes les « Persons »
- Quelles sont les noms de toutes les « Groups »
- Qui font partis de « badguys »
- Qui James Bond connaît-il ?
- Quelles informations peut-on savoir sur « James Bond » ?
- Qui travaillent au MI6 ?