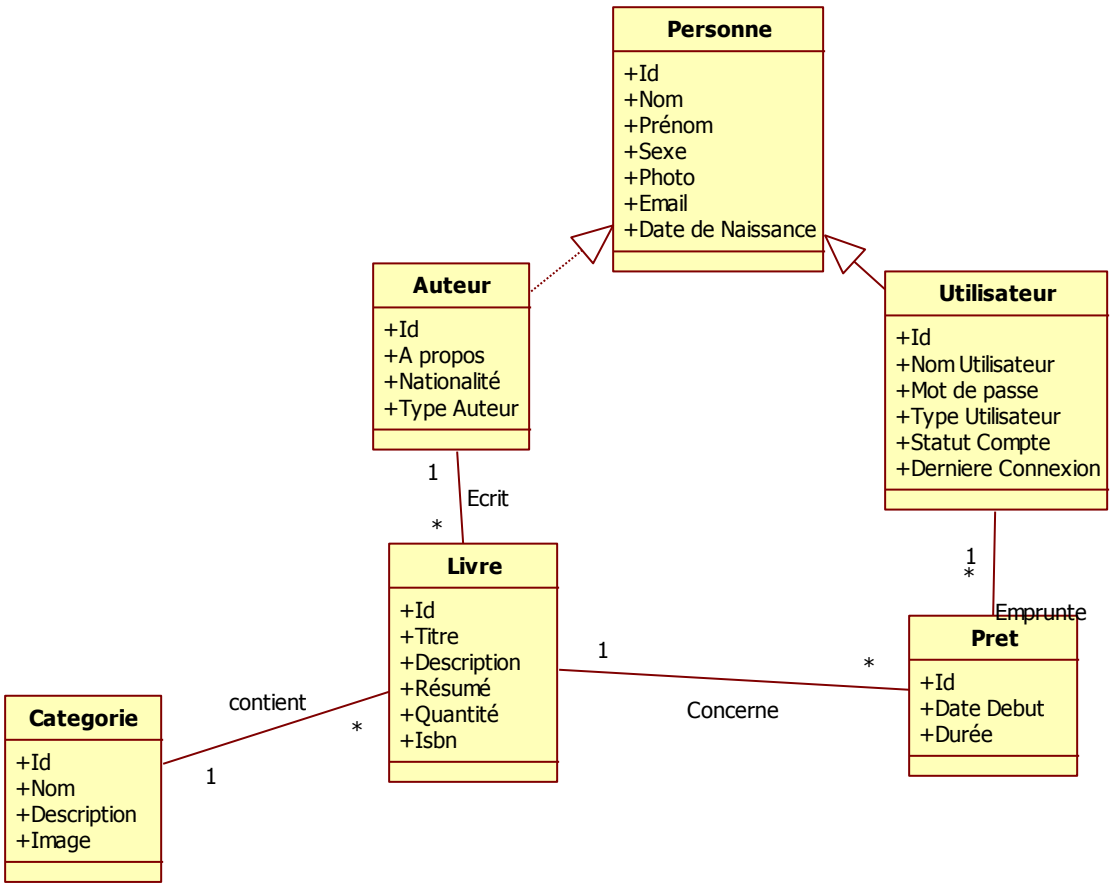
Gestion Bibliothèque

Résumé

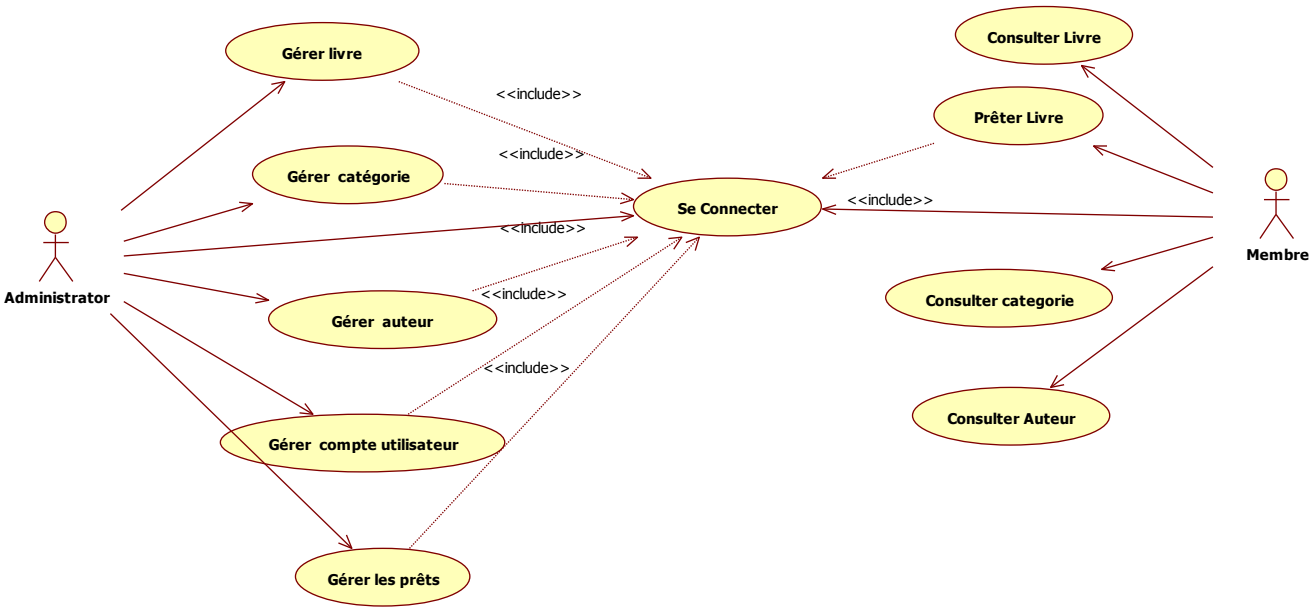
Ces travaux seront réalisés dans le cadre du cours Web Services Miage NTDP afin de permettre aux étudiants d'assimiler les concepts théoriques vus en cours. L'idée est de développer une application (réelle) et d'essayer d'appliquer les concepts du cours. Nous proposons le développement d'une application de gestion d'une bibliothèque dans laquelle les étudiants ajouteront un module web Service afin d'exposer les entités de l'application comme ressource, pouvant être (ré) utilisée par d'autres applications serveurs et principalement des applications clientes (HTML5, Android).

Contexte de l'application

1. Les discussions avec le client ont permis de modéliser le diagramme de classe suivant :



2. De ces discussions, nous avons pu dégager les cas d'utilisation présentés sur la figure suivante



Travaux à réaliser

1. Les travaux à réaliser dans le cadre de ces activités consistent à :
    a. Développer une application serveur permettant la gestion d'un modèle de données correspondant à celui de la figure 1
    b. Administrer ce modèle dans une base de données Relationnelles
    c. Exposer les entités de ce modèle dans un web service REST
    d. Développer un client HTML5 permettant de manipuler ce modèle depuis une interface web (ordinateurs, tablettes, Smartphones)
    e. Développer un client mobile Android permettant à un abonné de la bibliothèque de pouvoir consulter les livres disponibles et d'effectuer un prêt.

Contraintes

- L'application serveur doit être réalisée en langage Java en utilisant les EJBs
- Le service web doit être de type REST
- Le service doit supporter les formats JSON et XML

Partie 1 : Développement de l'application RESTFul

1. Le modèle de données

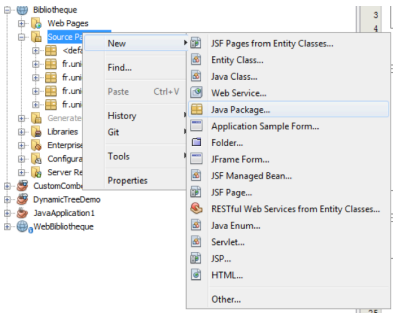
En ce qui nous concerne, ce module n'est pas une priorité. Nous supposons que vous avez les bases nécessaires vous permettant de le développer à partir des connaissances acquises dans le cours de J2EE.

Nous supposerons dans la suite de ce document que vous avez récupérez le projet source depuis github « https://github.com/eamosse/bibliotheque_ntdp.git »

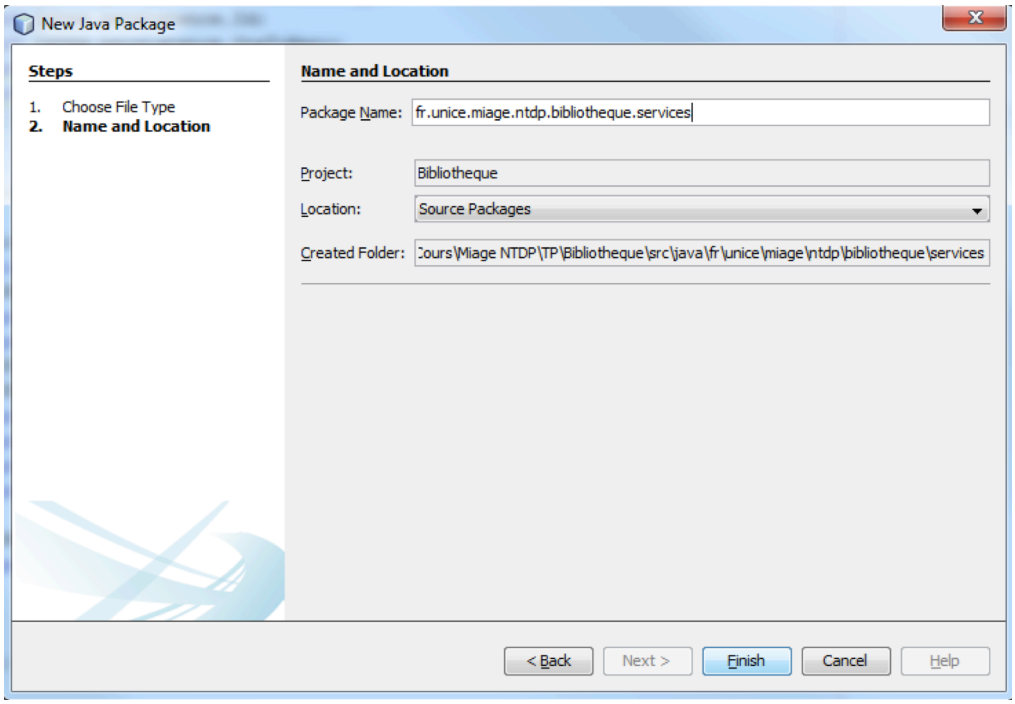
A. Exposer les entités comme ressources dans un web service REST

a. Dans le dossier Source Package de votre application « Bibliotheque **», ajoutez un nouveau package « fr.unice.miage.ntdp.bibliotheque.services »**

1. Faites **bouton doigt** sur le dossier **Source Packages ➔ New ➔ Java Package**



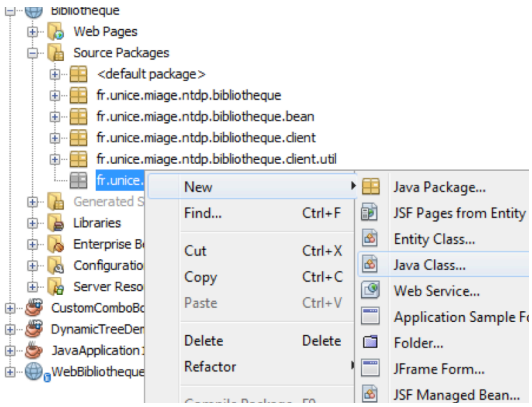
2. Entrez « **fr.unice.miage.ntdp.bibliotheque.services » ➔ Finish**



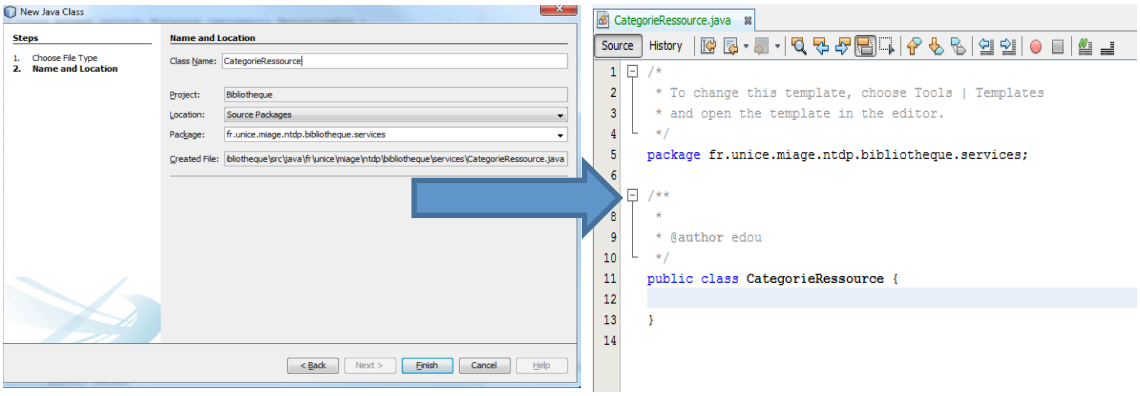
b. Exposer la classe Catégorie comme ressource

Comme nous l'avions vu en cours, JERSEY propose un ensemble d'annotation permettant d'exposer une classe Java comme ressource dans une application RESTFull ! Il suffit d'ajouter l'annotation @Path sur le classe en lui donnant un nom statique en paramètre.

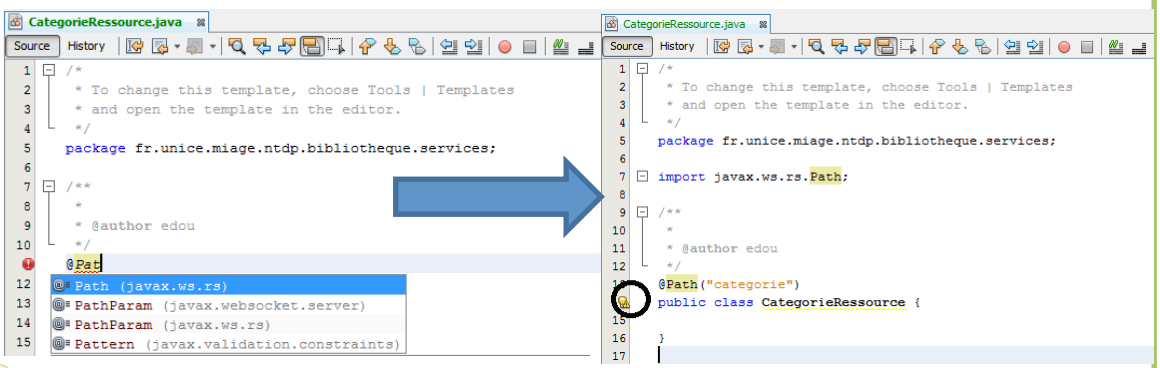
1. Ajouter une nouvelle classe dans le package « service » créé récemment. Faites **bouton doigt service ➔ New ➔ JavaClass !**



2. Dans l'écran qui s'affiche, **Donnez une valeur au champ Class Name** (Mettez **CategorieRessource** par exemple) ➔ **Finish**



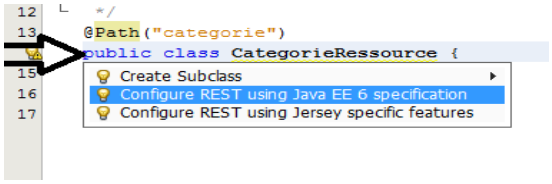
3. Ajoutez une **annotation @Path** sur la classe « **CategorieRessource** » ! N'oubliez pas de préciser le nom en paramètre à l'annotation.



Avez-vous remarqué la petite ampoule à droite de la classe ?

Elle vous signale que la classe qui expose toutes les ressources de votre projet n'a pas été créée. En effet, en lieu et place d'un fichier de configuration, JERSEY utilise une classe qui permet de gérer la configuration de l'application : renseigner les classes exposées comme ressources, le contexte de base de ressources etc.

4. **Cliquez** sur la petite **ampoule** ➔ Choisissez **Create REST using Java EE 6 specification**



Remarquez l'ajout d'un nouveau Package dans le projet « **org.netbeans.rest.application.config** », vous pouvez le laisser ainsi où le renommer, le nom n'a pas d'importance ! Remarquez egalement la création d'une nouvelle classe « **ApplicationConfig** », vous pouvez la renommer si ça vous tente !

A partir de Java EE 6, cette classe est proposée comme alternative aux paramètres du service dans le fichier de configuration « web.xml ».

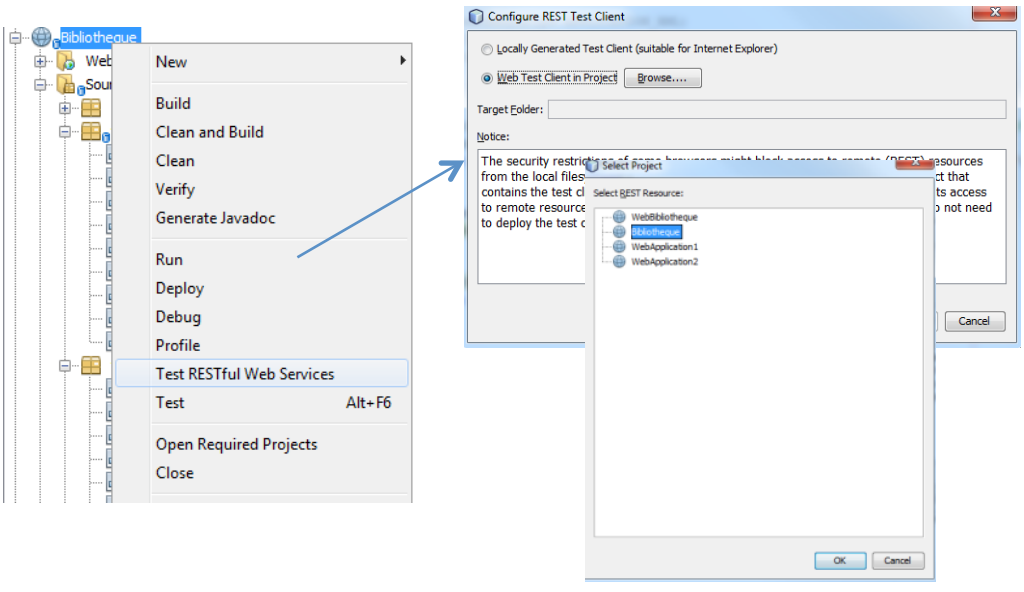
Ouvrez la classe Application Config dans votre éditeur

- ⇨ Remarquez l'annotation **@javax.ws.rs.ApplicationPath("webresources")**
- ⇨ Observez bien les méthodes de cette classe, elles devraient vous réveiller votre mémoire sur certaines choses qu'on a vues en cours.

5. Ajoutez une méthode qui répond à une requête HTTP/GET indiquant que la ressource demandée existe.

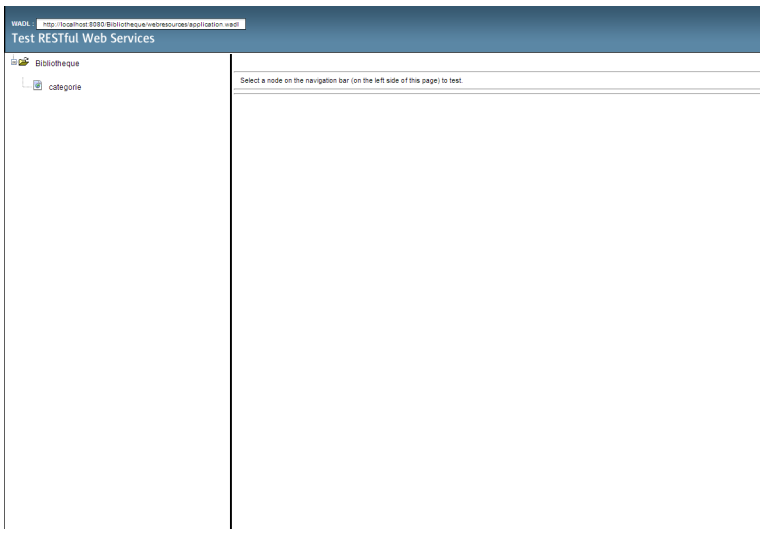
```
@GET
  @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public String hello() {
      return "La ressource demandée existe!";
  }
```

6. Testez ! Faites **bouton doigt** sur votre **projet** et dans le menu qui s'affiche choisissez **Test RestFul Web Service**
Dans la fenêtre qui s'affiche sélectionnez **Web Test Client in Project** ➔ **Browse** ➔ Sélectionnez **Bibliotheque** ➔ **OK** ➔ **OK**



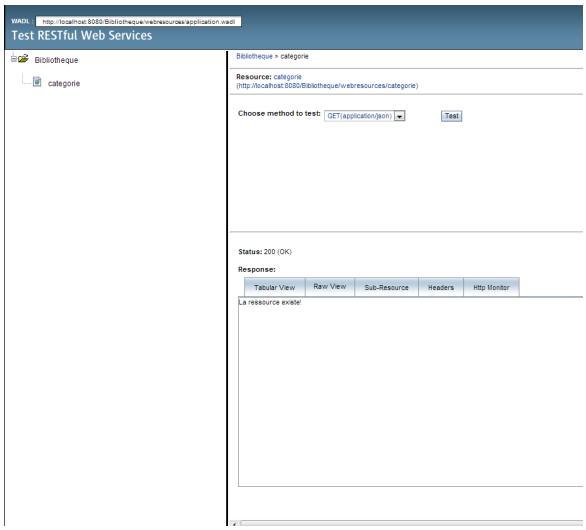
Après cette étape, Vous devriez voir de nouveaux fichiers s'ajouter à votre projet ! Cet outil ajoute à votre application web un ensemble de fichiers statiques (html, css, javascript) qui vous permettront de tester votre service via une interface web !

7. Si tout s'est bien passé, Netbeans a dû lancer automatiquement votre navigateur avec un lien vers le document de test ! Si ce n'est pas le cas appelez au secours.



8. Dans le panel de gauche de votre écran vous devriez avoir la liste des ressources qui ont été exposées dans votre application

⇨ Cliquez sur **categorie**, dans le panel de droite **choisissez un format de représentation** correspondant à l'un des formats de produit par la méthode, puis cliquez sur **test.** Dans la section réponse, vous devriez voir la donné renvoyée par la méthode s'afficher.



9. Modifiez la classe **CategorieRessource** pour y ajouter les méthodes correspondant aux opérations CRUD

Le projet dispose d'une interface permettant de persister les objets dans une base de données relationnelle, en utilisant les EJBs. Réutilisez cette classe afin de simplifier l'accès à la base de données

1. Faites hériter la classe **CategorieRessource** de **AbstractFacade** (**package bean**)

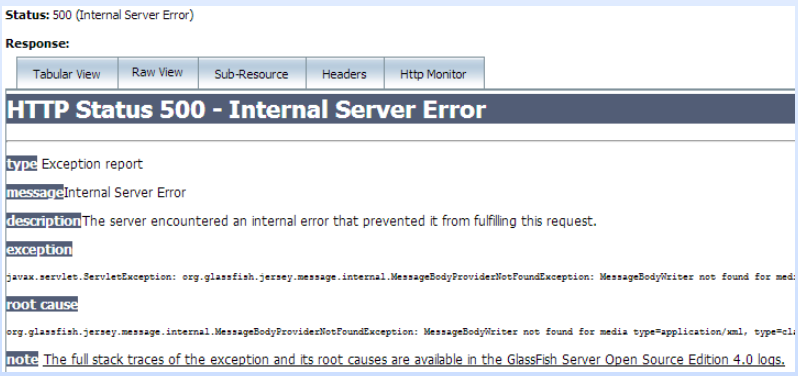
```
@Path("categorie")
public class CategorieRessource extends AbstractFacade<Categorie> {

      public CategorieRessource() {
          super(Categorie.class);
      }
.....
}
```

2. **Ajoutez** une méthode **List** qui renvoie la liste des catégories

```
@GET
@Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
public List<Categorie> list(){
     return super.findAll();
}
```

3. Refaites les étapes 6, 7 et 8 et observez les résultats obtenus

- Utilisez les interfaces de l'application web pour ajouter de nouvelles données dans la base
- Normalement, la représentation au format JSON se passera sans grandes difficultés
- Par contre, le format xml risque de vous poser quelques soucis, la plupart d'entre vous auront une erreur semblable à celle-ci :



- Ne vous en affolez pas, il vous manque un petit quelques choses
  En effet, si vous vous souvenez dans la classe ApplicationConfig, vous avez chargé une classe dans les ressources
  « Class.forName("org.glassfish.jersey.jackson.JacksonFeature"); », cette librairie de classe déjà intégrée à glassfish se charge du mapping objet➔ JSON, JSON➔ Objet.
  Pour le XML, il faut ajouter une annotation spécifique sur les entités à exposer.
  Si vous voulez qu'une classe (entité) soit serialisable en xml, il suffit d'ajouter l'annotation « **@XmlRootElement** » du package « **javax.xml.bind.annotation** »

4. Ajoutez une méthode create permettant d'ajouter une nouvelle catégorie dans la base

Cette méthode doit prendre en paramètre un objet de type catégorie et renvoyer un message de confirmation ou d'échec en fonction du résultat de la transaction avec la base de données.

- La méthode acceptera des données sous deux formats (XML ou JSON)
- La méthode doit renvoyer une réponse soit en XML ou en JSON

A ce stade vous devriez vous demander comment attendre des données en JSON ou en XML alors que la méthode prend en paramètre un objet de type catégorie!!!!

En effet, le client à l'appel de la méthode doit sérialiser ses données dans l'un des formats supportés par la méthode (XML ou JSON), mais avant l'appel à la méthode JERSEY s'occupe de la dé-sérialisation à notre place. Ainsi on peut vitre tranquillement dans notre petit monde ou tout est objet.

6. Exercices (à rendre à la fin du cours)

- Complétez la ressource « catégorie » en y ajoutant les méthodes suivantes :
- **Update -->** Modifier une catégorie existante
- **Delete-->** Supprimer une catégorie identifiée par son id
- **Count -->** Renvoie le nombre de catégories existantes dans la base
- **FindByRange -->** Renvoie les catégories entre un id(min) et un id(max)
- Utilisez l'interface web de test pour tester toutes vos méthodes (Ajout, Suppression, Mise à jour

7. Exercices (à remettre le vendredi 28/11/2014)

- Exposez les autres entités du modèle comme ressources
- Ajoutez une méthode dans la ressource, permettant de retrouver tous les livres d'une catégorie
- Ajoutez une méthode permettant de lister tous les prêts relatifs à un livre
- Ajoutez une méthode permettant de retrouver tous les prêts relatifs à un utilisateur donné
- Ajoutez une méthode permettant de changer le statut d'un prêt
- Ajoutez une méthode permettant de désactiver/Activer un compte utilisateur

Fin de la partie 1