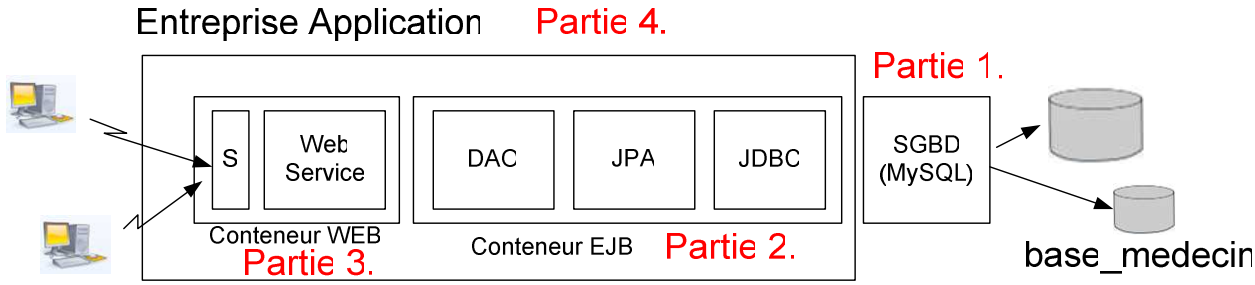
Création d'une application JEE

Rédacteurs : Alexandre Baillif, Philippe Lacomme et Raksmey Phan
Date : juillet 2010
Avertissement : ce document est une reprise d'une partie d'un document écrit par Serge Tahé. En particulier la base de données utilisée.

Ce support de cours concerne la mise en place d'une application JEE avec un client.

La présentation comprend 5 parties :

Partie 1. Création d'une base de données sous MySQL.
Partie 2. Création d'un conteneur EJB
Partie 3. Création un conteneur Web pour accéder à l'EJB
Partie 4. Création d'une « Entreprise Application »
Partie 5. Création d'un client web.



Partie 5.

La base de données s'appellera **base_medecin**.

JDBC : cette couche gère la connexion avec la (ou les) base(s) de données. Ici on utilisera la notion de pool de connexion. Un pool de connexion est un ensemble de connexions avec la base de données déjà instanciées. Cela permet aux requêtes de s'exécuter plus rapidement. On peut venir connecter plusieurs couches JPA sur la couche JDBC si nécessaire.

JPA : la couche JPA (Java Persistence Annotation) est une couche d'abstraction de la couche JDBC. Elle permet notamment de faire du Mapping Relationnel-Objet (ORM, Object-Relationnal Mapping en anglais) qui consiste à modéliser la base de données sous forme d'objets pour une manipulation plus simple à travers le code Java (requêtes pré-écrites, gestion des liens entre les tables,…). Généralement la couche JPA contient une classe (entité) par table, des contrôleurs (fonctions de base implémentées) et des gestionnaires d'exceptions.

DAO : Cette couche représente l'intelligence de l'application. Elle est composée d'un ensemble d'interfaces locales (local) et distantes (remote). Les DAO (Data Access Object) permettent d'accéder aux objets et proposent des méthodes de CRUD (Create, Read, Update, Delete). Un EJB (Entreprise Java Bean) sera piloté à partir d'une autre application distante ou locale (client EJB).

Web Services : Cette couche a pour but de définir des services qui pourront être appelés selon le protocole SOAP. Ainsi les informations pourront circuler entre les applications sous forme de messages XML. Cela peut servir à faire communiquer deux applications qui peuvent être codées dans deux langages différents, en local ou à distance.

Partie 1. Création d'une base de données MySQL

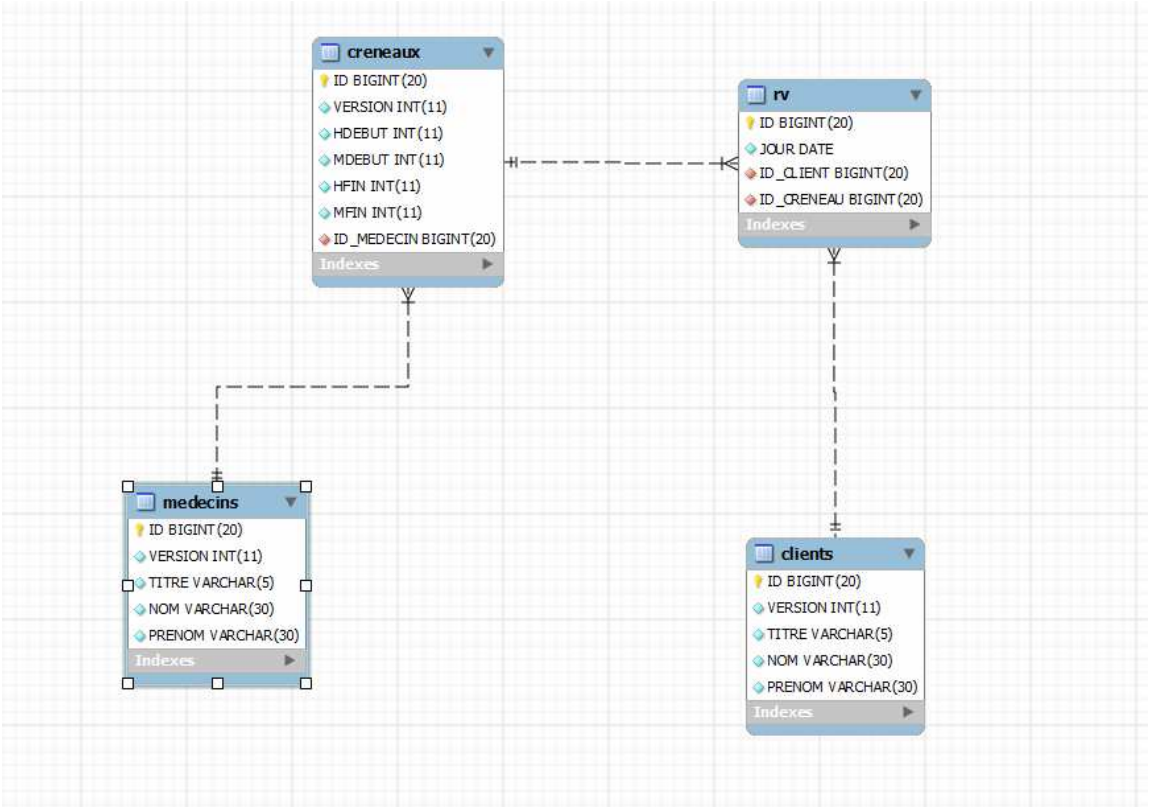
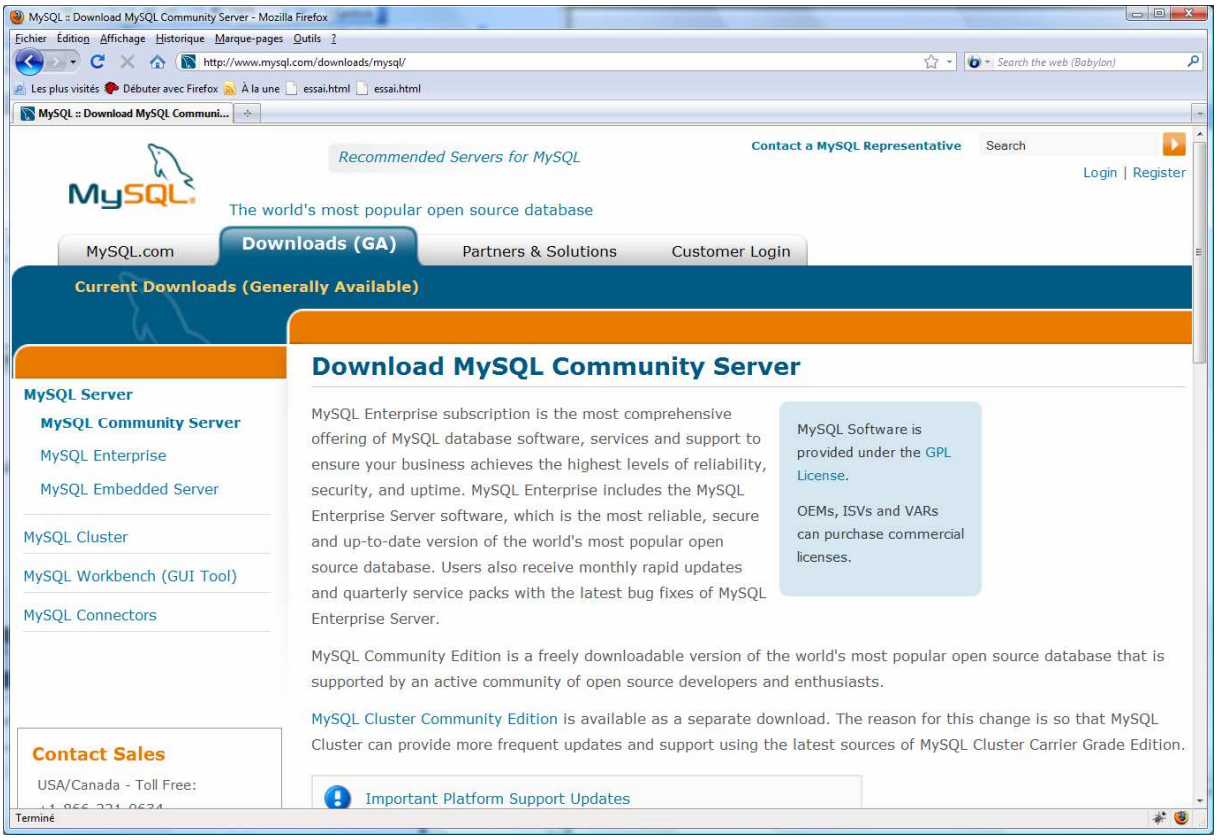


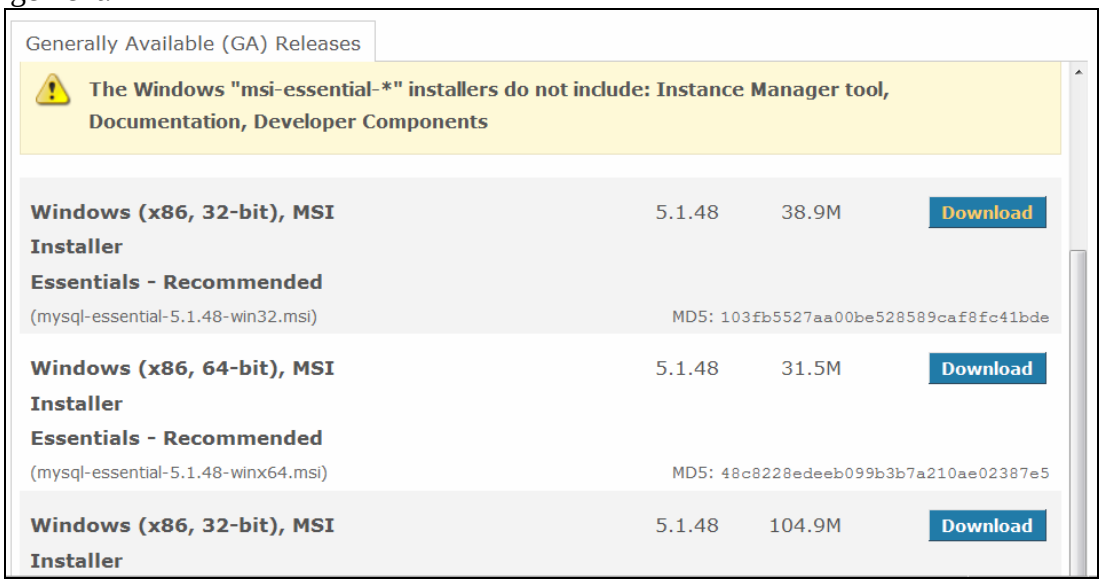
Figure 1: Base de donnée finale

1) Téléchargement de MySQL

Nous utilisons une base de données MySQL.
Site : http://www.mysql.com/
Utiliser la section **Download**.

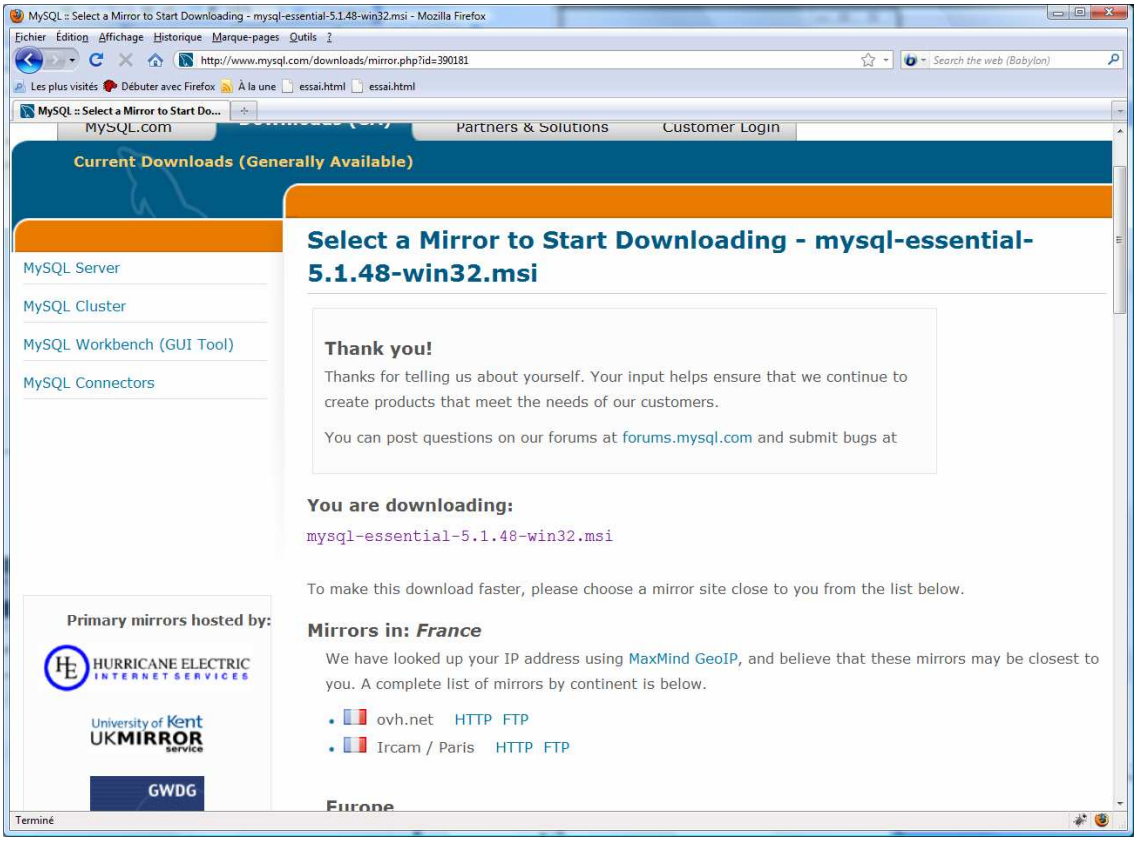


Choisir ensuite votre système d'exploitation. Par exemple Windows 32 bits et lancer le téléchargement.

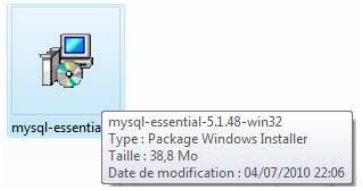


Après un questionnaire (un peu long ☹) vous demandant de vous identifier, vous pourrez accéder à la page de téléchargement.

Ou vous pouvez cliquer sur le lien « No thanks, just take me to the downloads! »

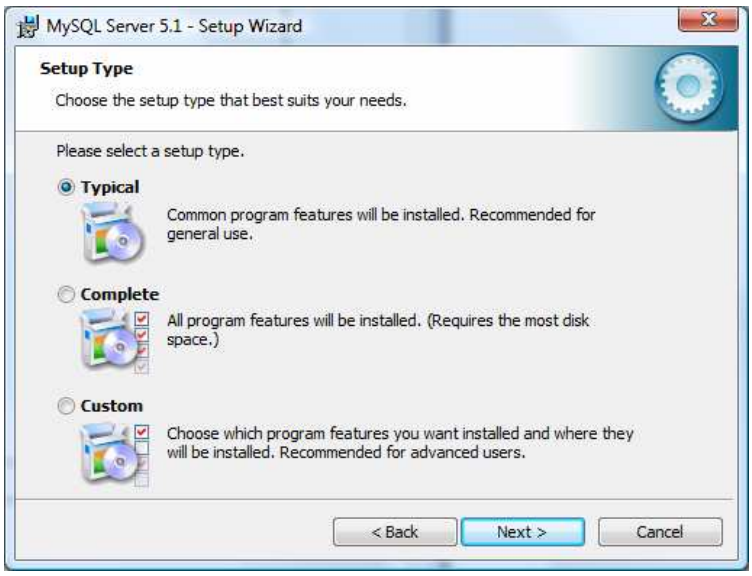


Le fichier télécharge se présente comme suit :

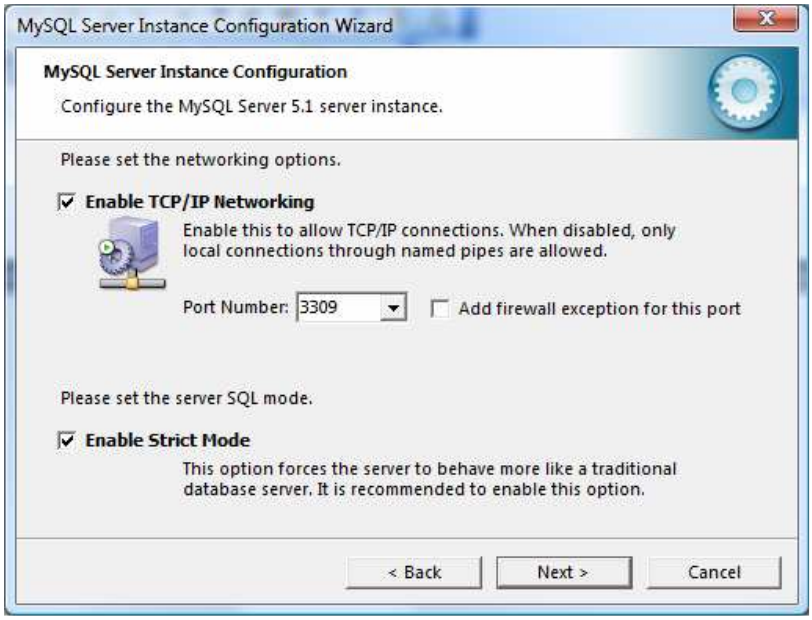


1.2. Installation de MySQL

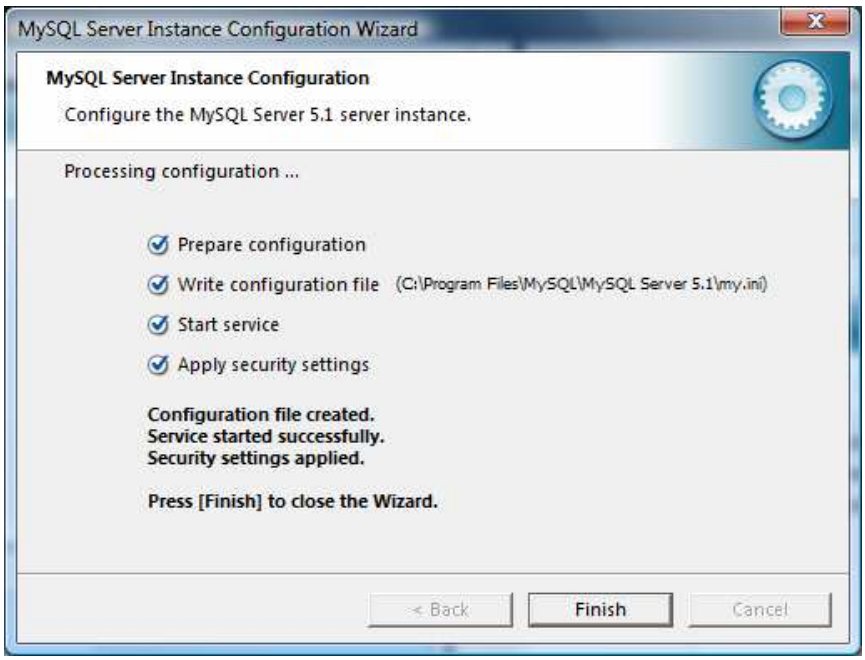
Conserver les réglages par défaut et valider les différents écrans d'installation.
Utiliser le compte *root* et le mot de passe *admin*.



Remarquons que pendant l'installation, nous avons accès au numéro de port (par défaut 3309).

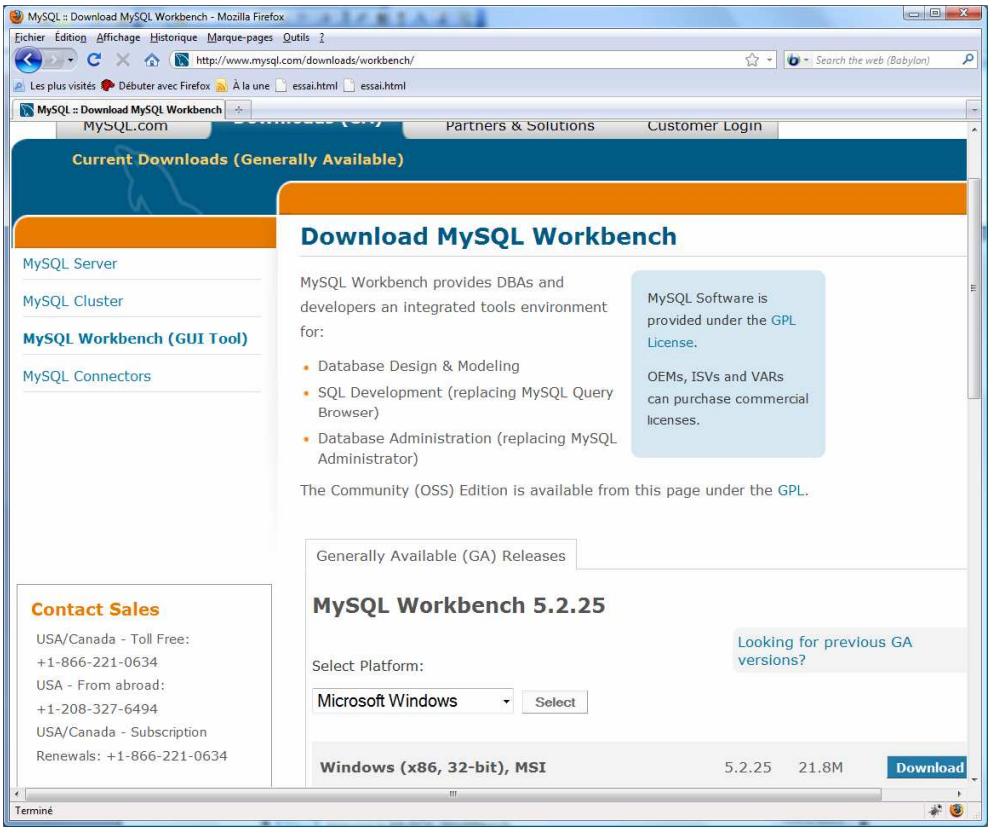


Si l'installation se passe bien elle devrait se terminer par l'écran qui suit :



1.3. Téléchargement et Installation de MySQL WorkBench

Cet outil n'est pas indispensable mais très efficace et permet de manipuler MySQL de manière très simple. Il est vivement recommandé de l'installer. Cette interface graphique est en fait une couche de manipulation de MySQL.

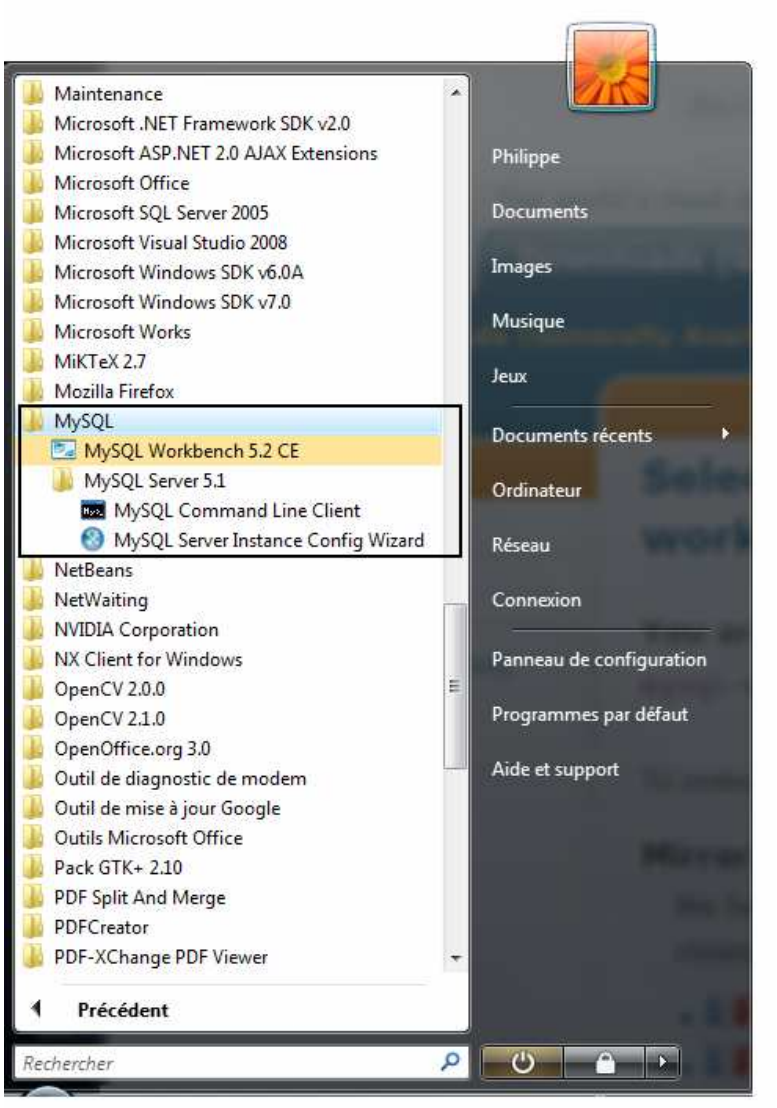


Lancer l'installation

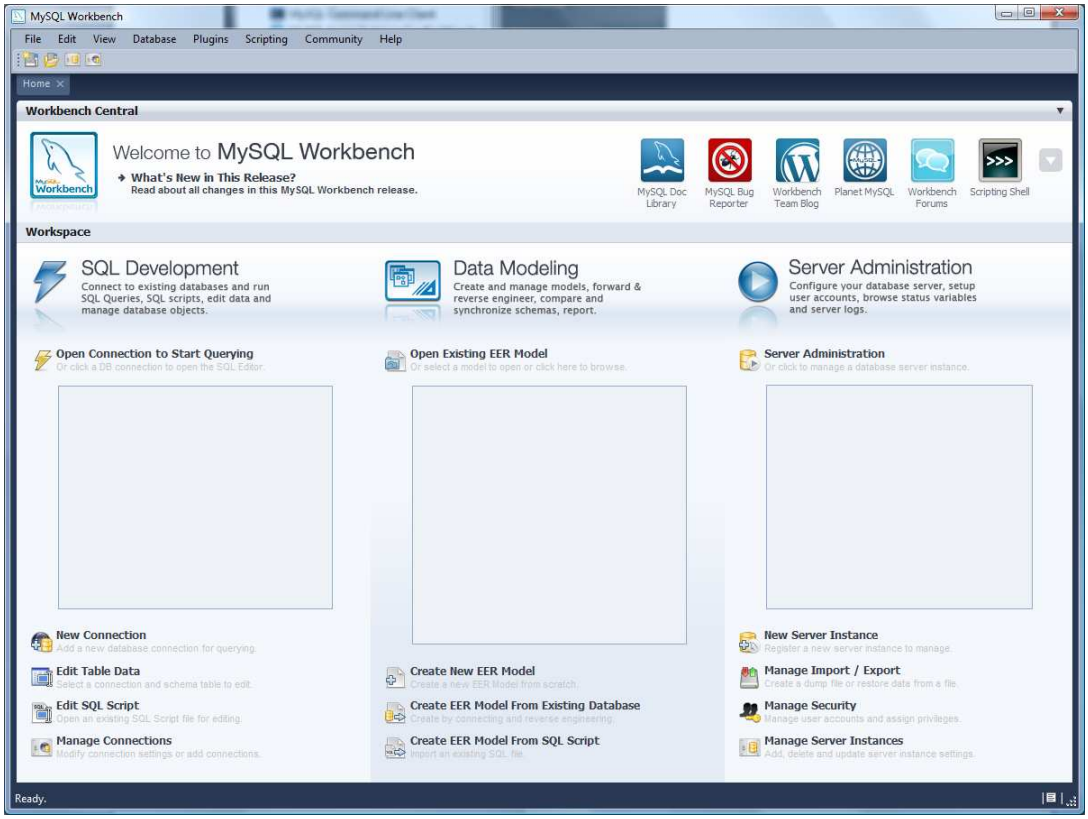
1.4. Ensemble des outils MySQL

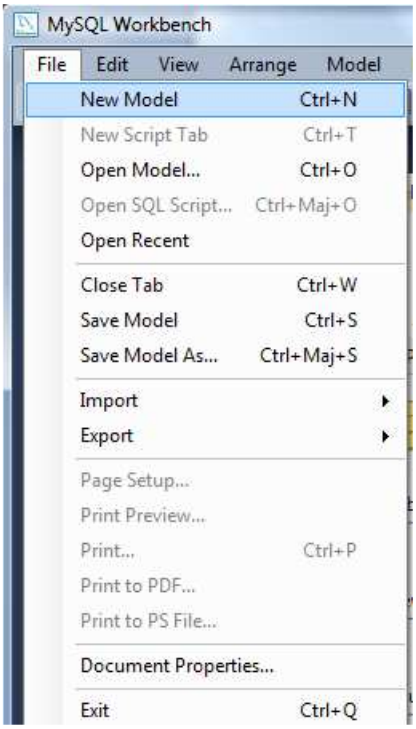
Dans le menu Démarrer, dans le sous-menu MySQL se trouvent :

- WorkBench
- MySQL Serveur.



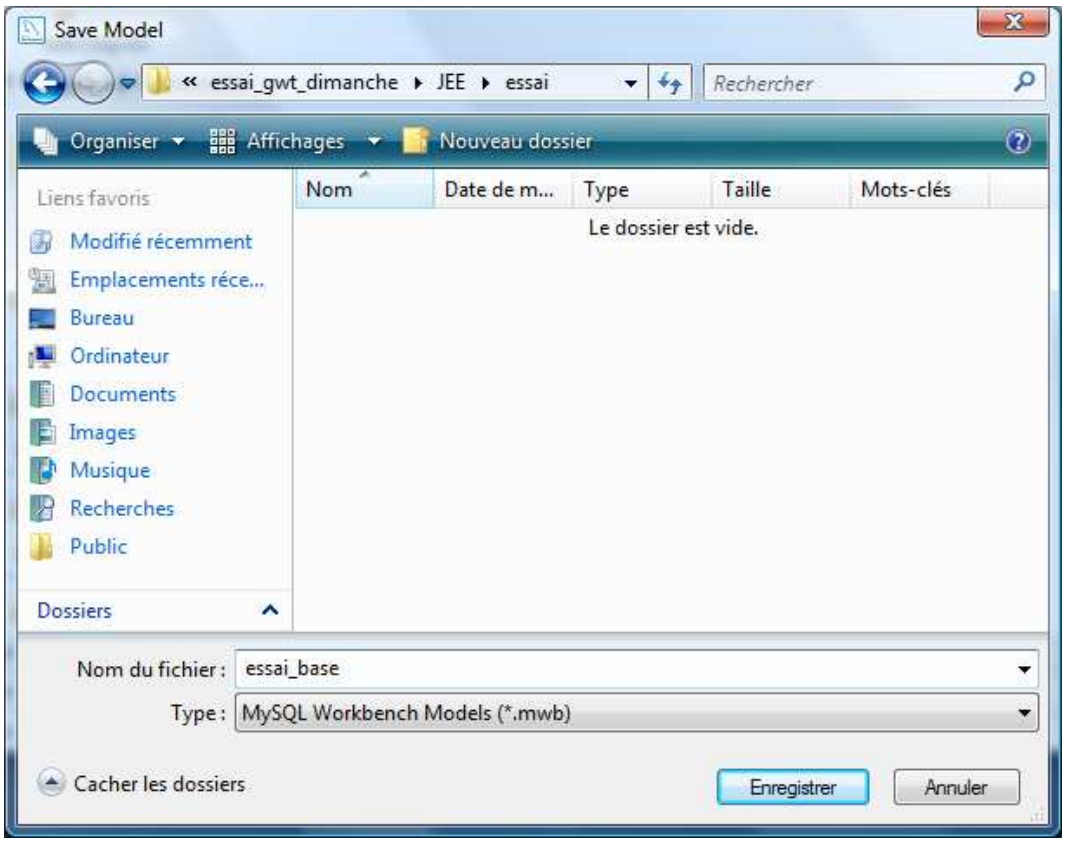
1.4. Création d'une base de données

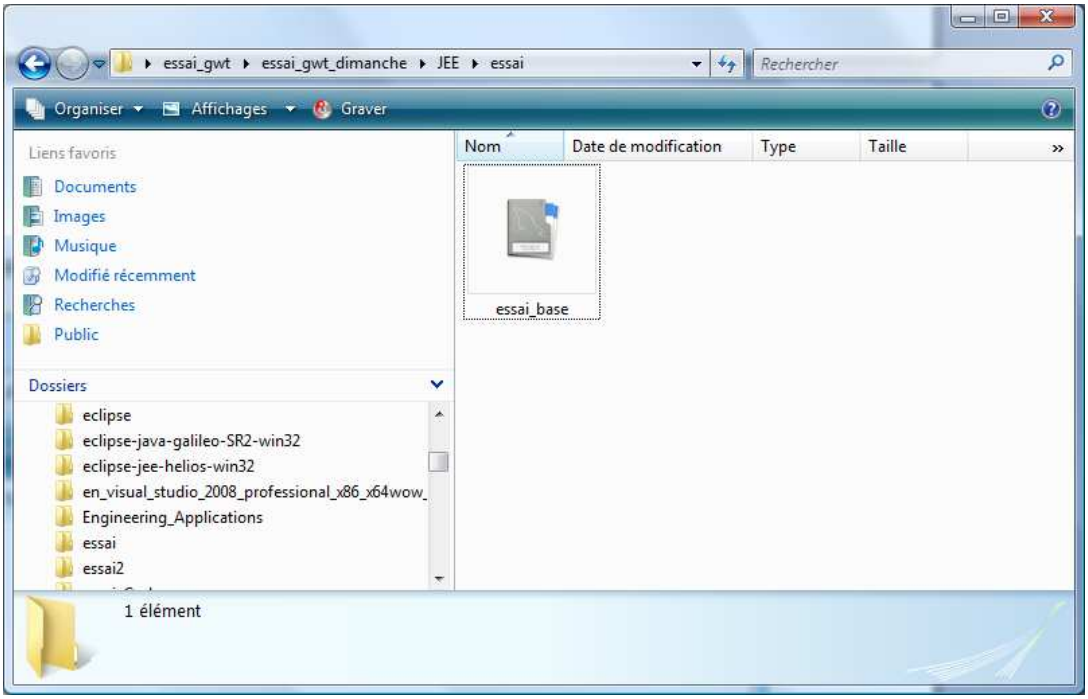




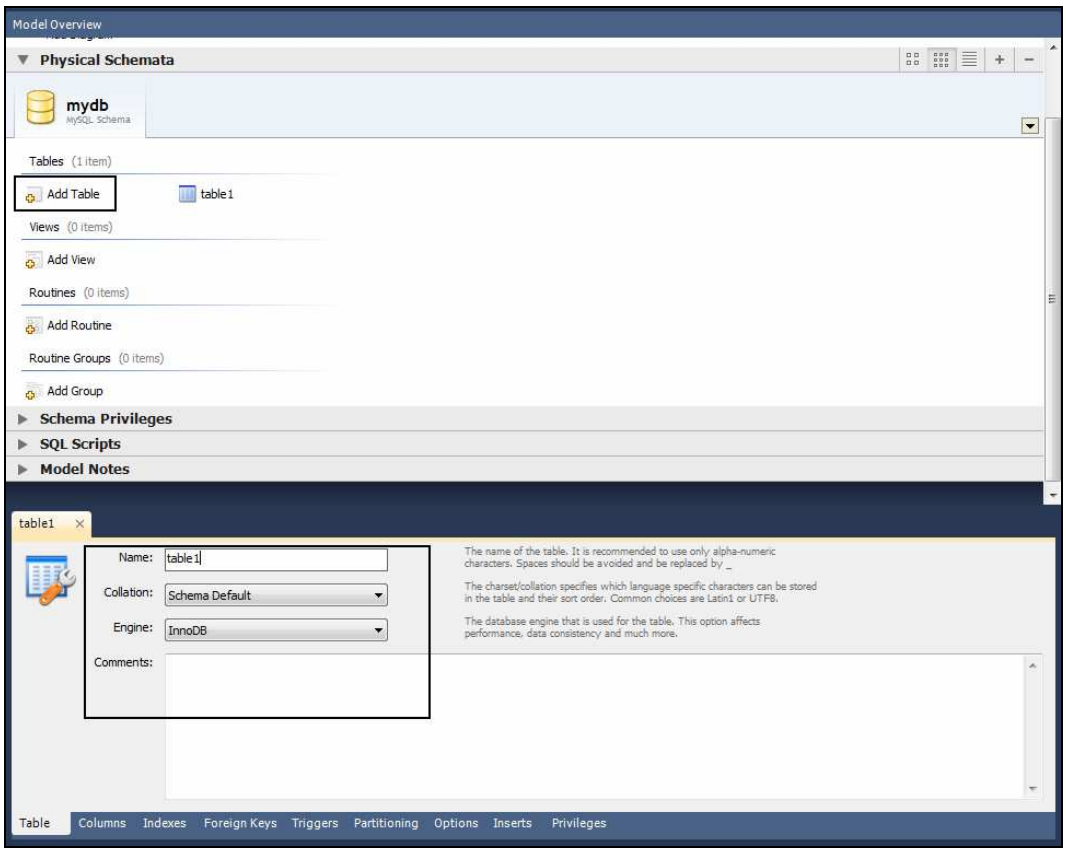
Créer une nouvelle base en utilisant : File / New Model.

Faire « Enregistrer Sous » et choisir un répertoire :

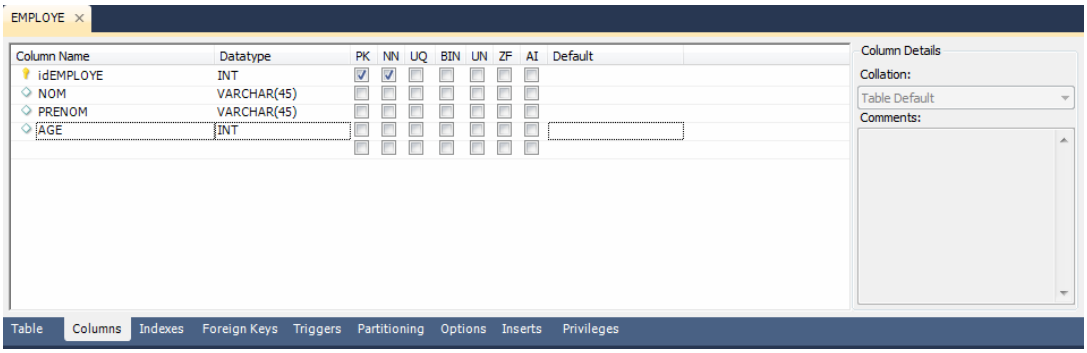


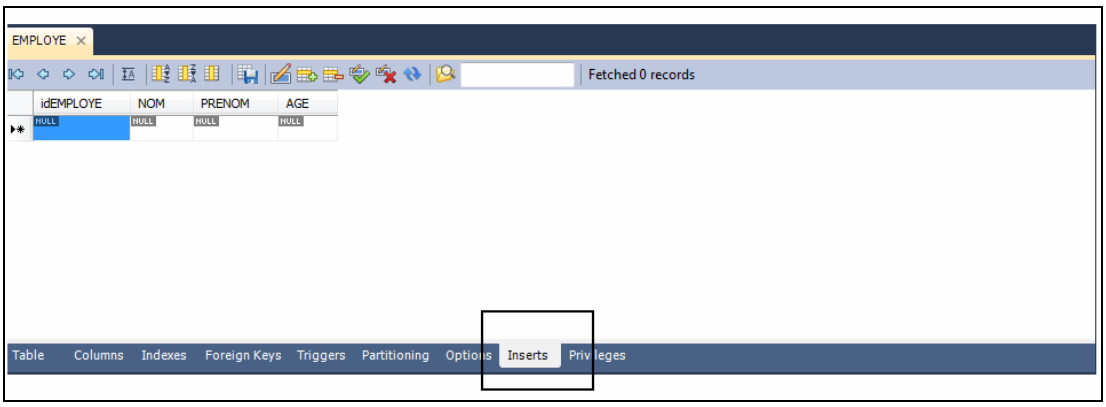


En cliquant sur « **Add Table** » on peut ajouter une table qui par défaut porte le nom table1.



Modifions ensuite le nom de la table en EMPLOYE et validons. Nous pouvons ensuite facilement ajouter les champs NOM, PRENOM et AGE.

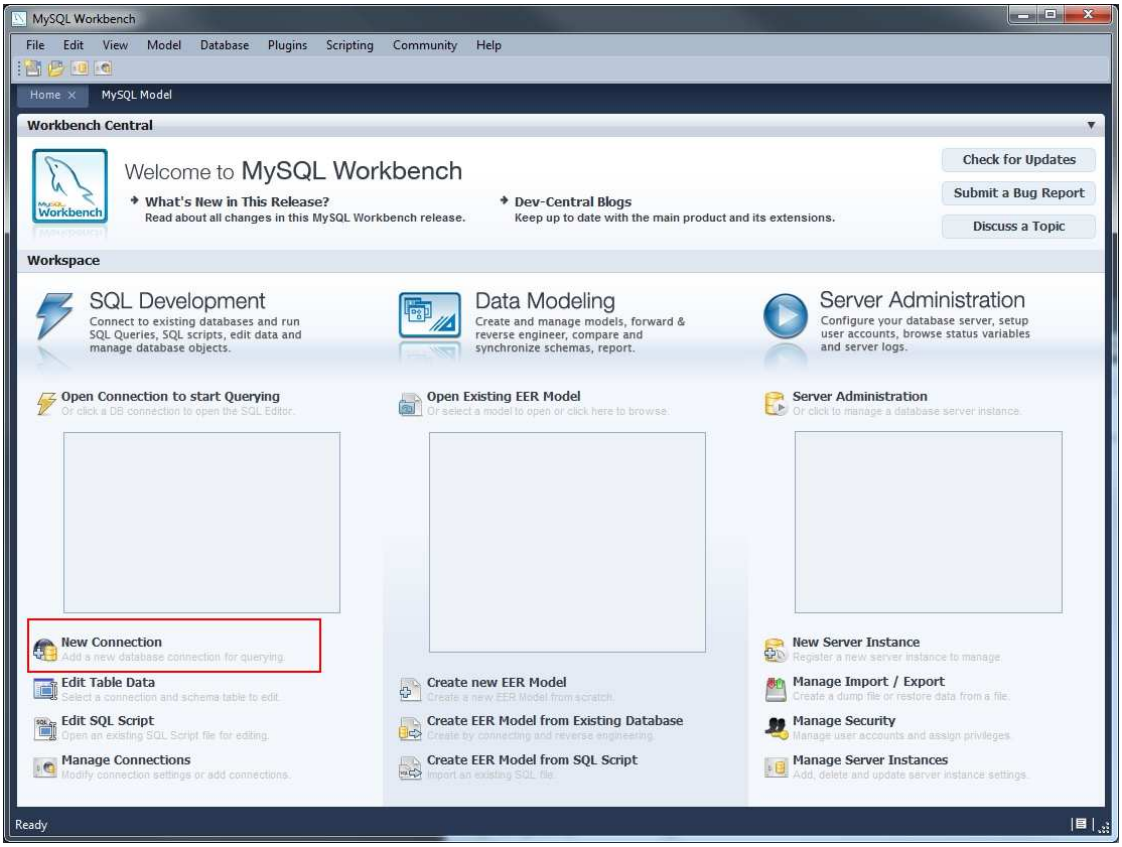




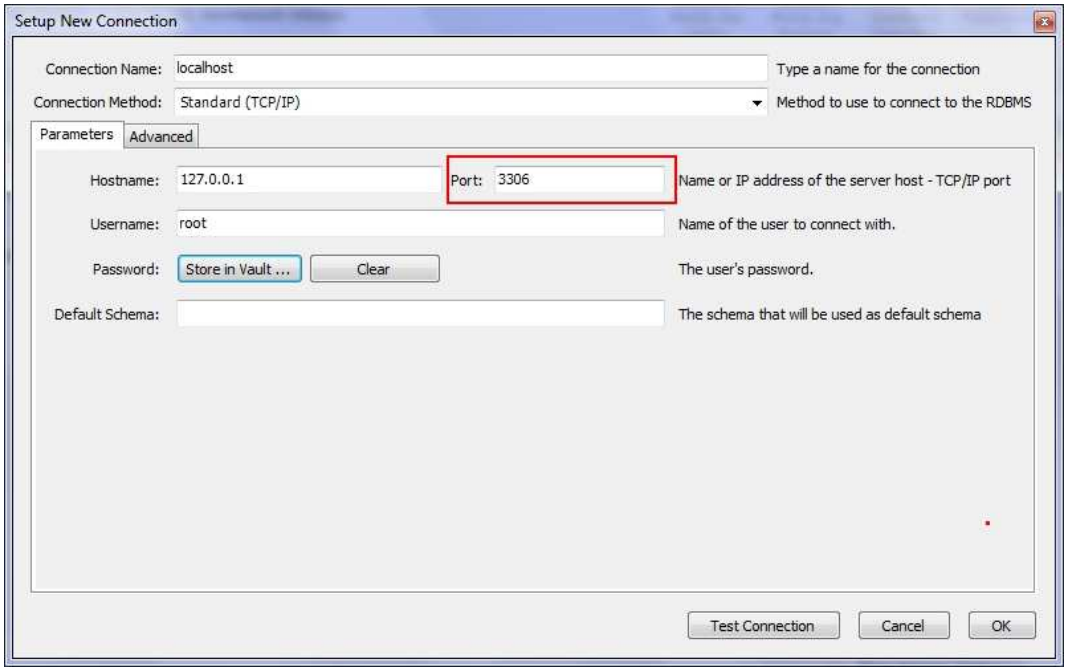
Sauvegarder le modèle.

1.5. Création d'une base de données à l'aide d'un script

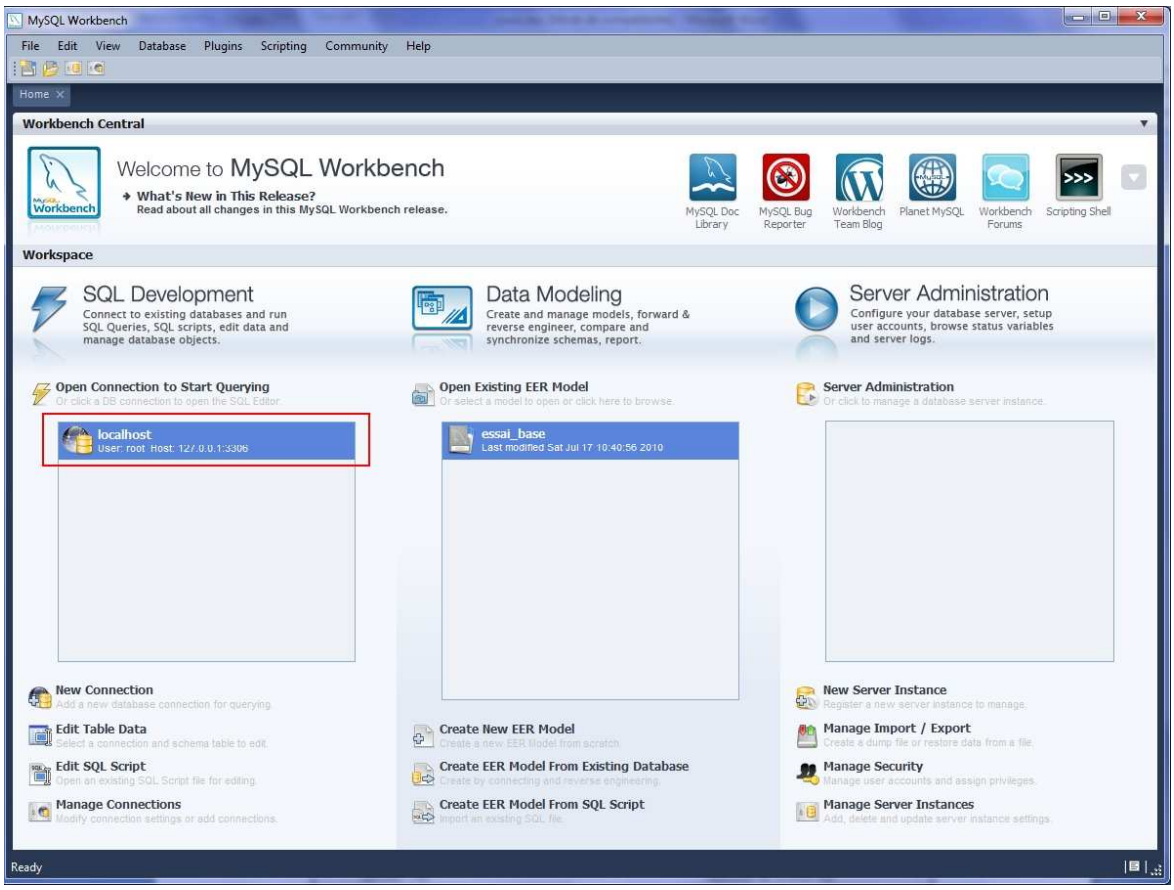
Revenez dans l'onglet « Home ». Faites « New Connection ».



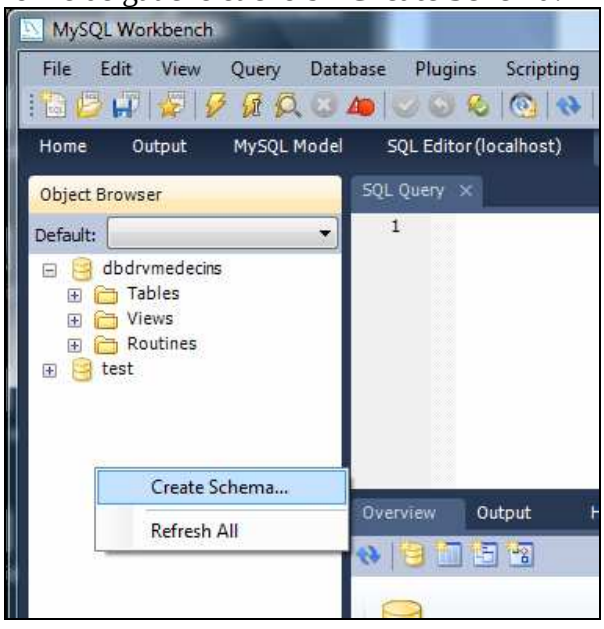
Utilisez le numéro de port par défaut donné lors de l'installation de MySQL (3306, 3309,…). Puis tester la connexion.



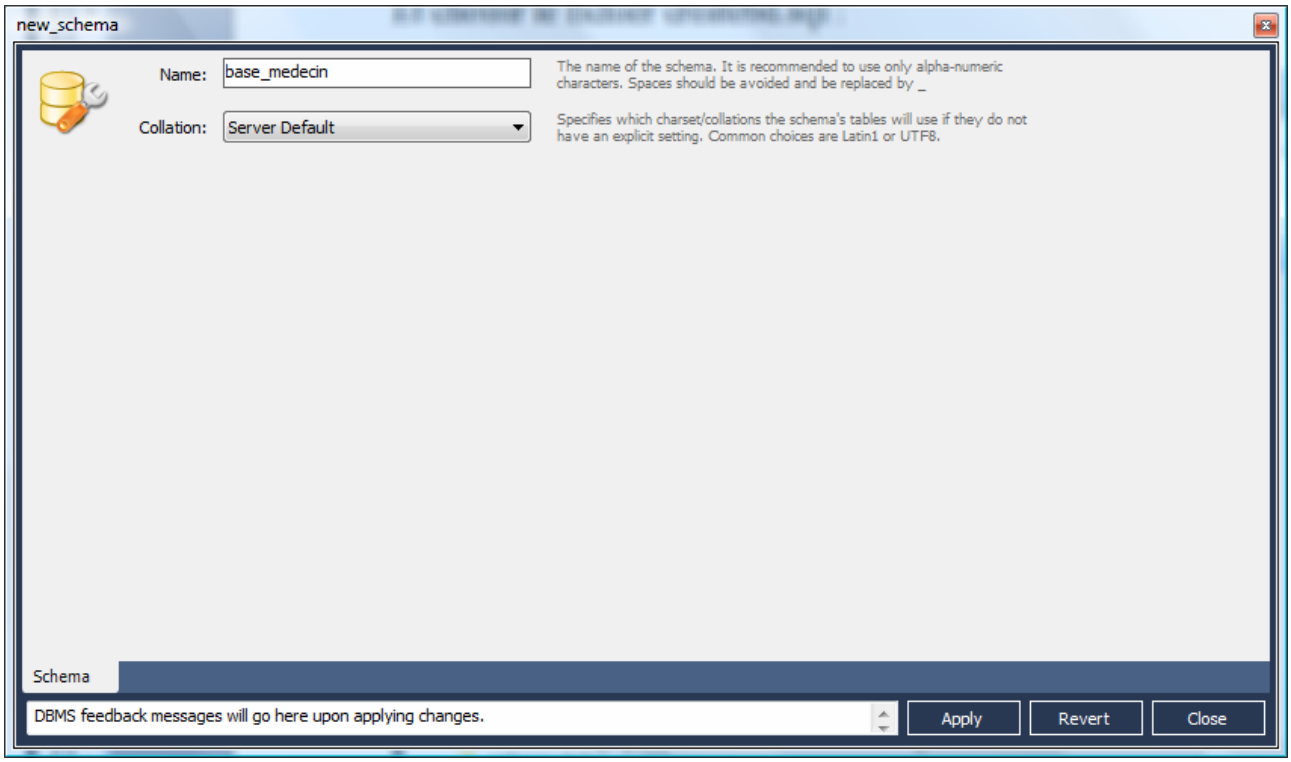
Au retour sur l'écran principal, double-cliquez sur « localhost ». Une nouvelle fenêtre de « SQL Query » s'ouvre.



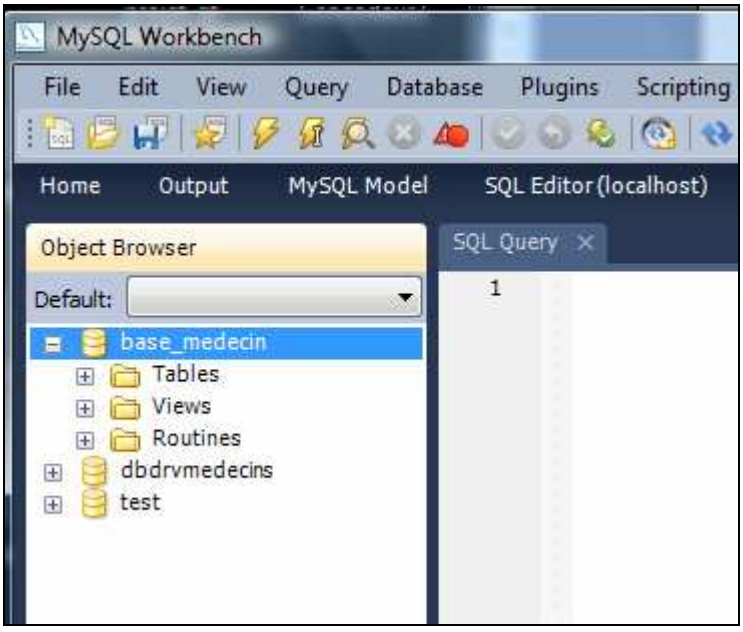
Faire un clic droit sur la colonne de gauche et choisir **Create Schema**.



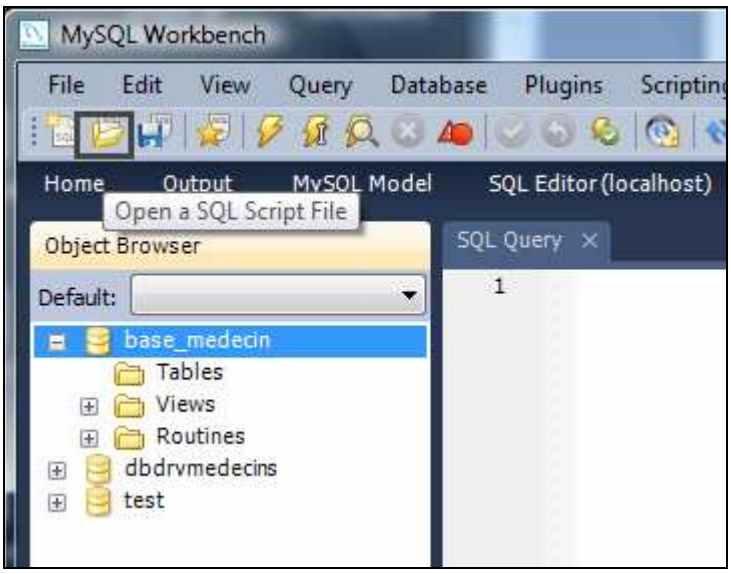
Donner un nom à la base de données : par exemple **base_medecin**.



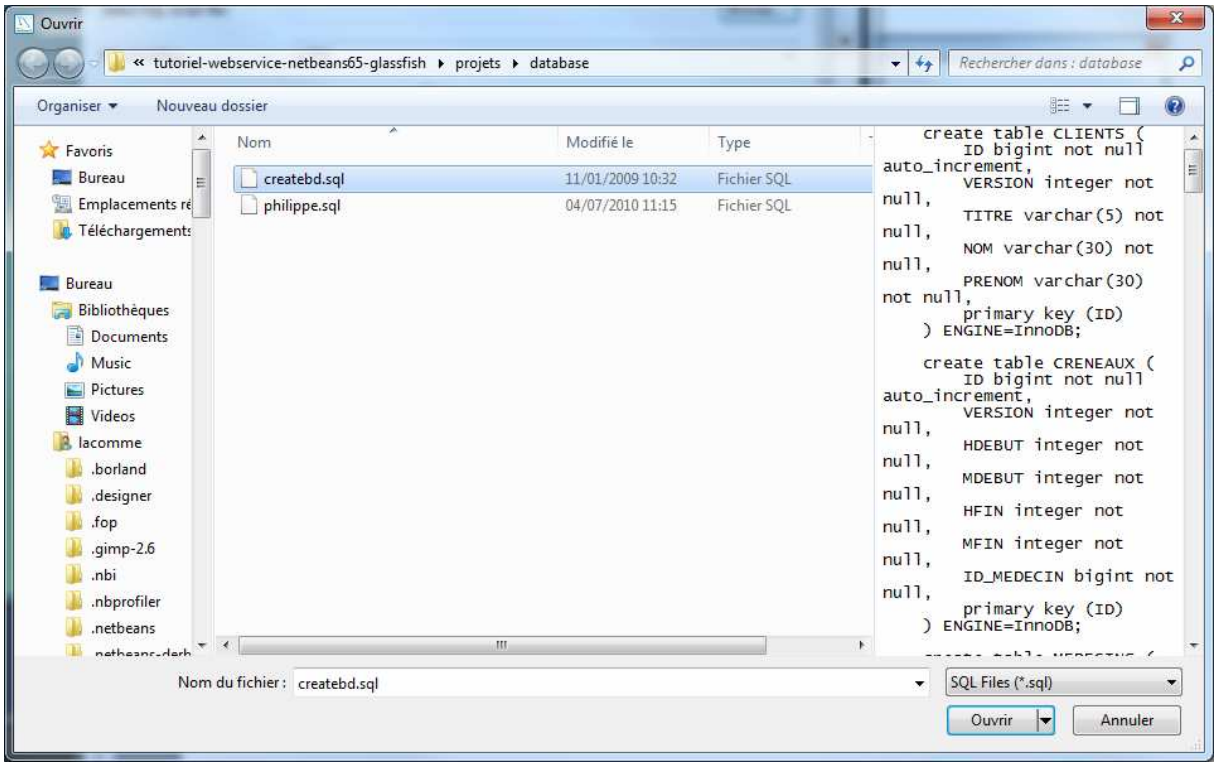
Valider les différentes étapes. La **base_medecin** apparaît ensuite dans la colonne de gauche.

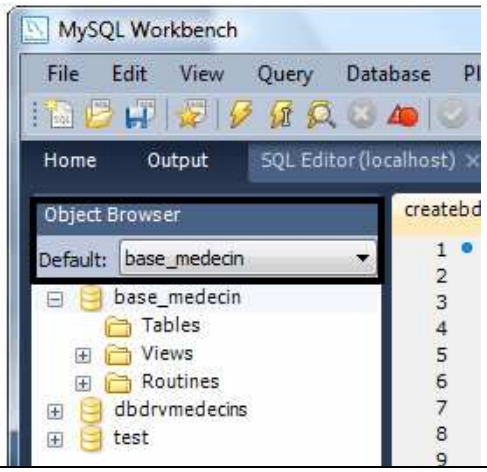
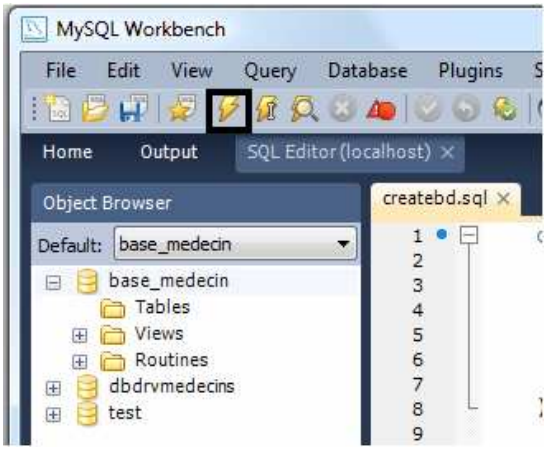


En utilisant la deuxième icône de la barre du haut, ouvrir un script.



Et choisir le fichier createbd.sql (cf. Fichier joint) :



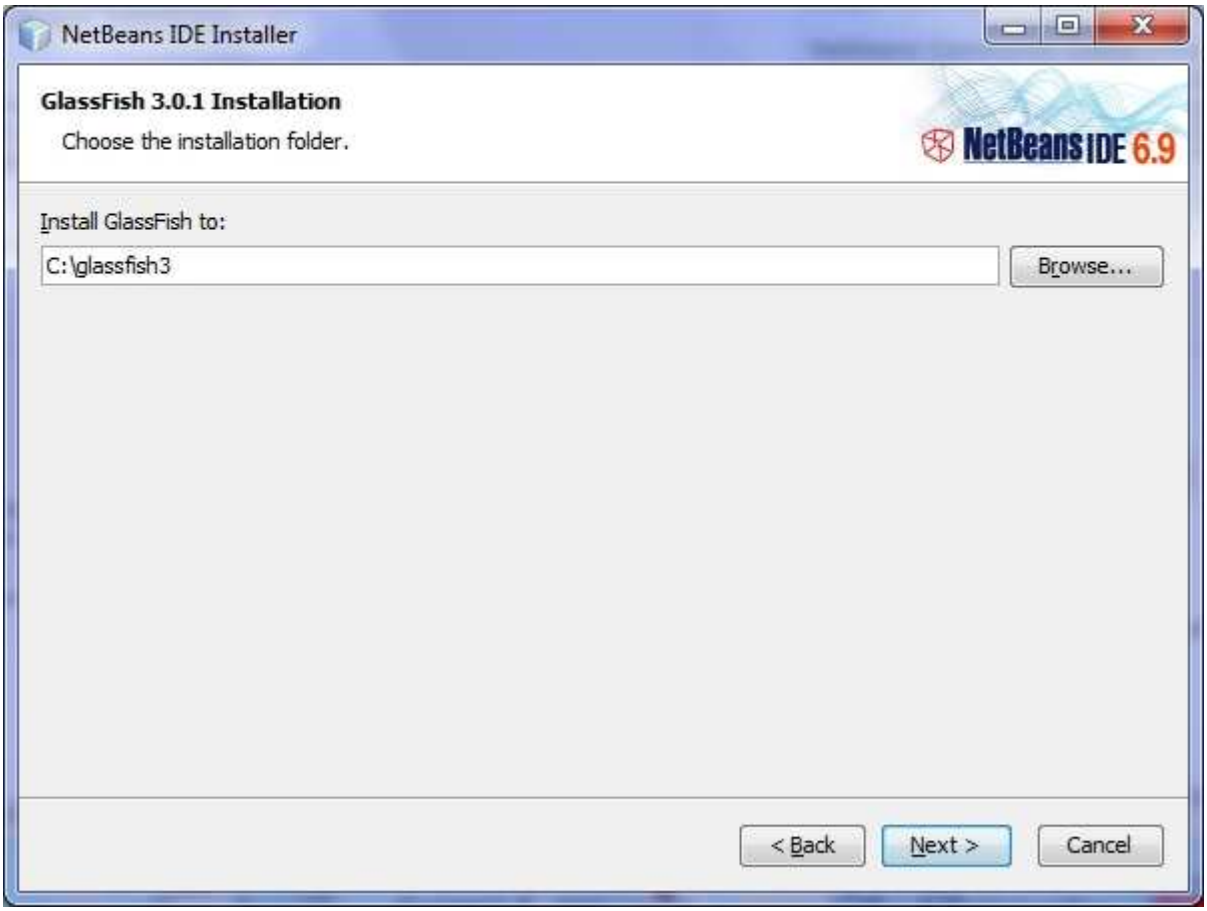
Assurez-vous d'avoir sélectionner **base_medecins** dans la partie **Default**. 	Et finalement **exécuter** le script. 

1.6) Télécharger et Installer Netbeans

Avant de télécharger Netbeans, il est important d'installer un JDK sur sa machine. Il s'agit d'un ensemble de librairie Java pour le développement.
Pour ce tutorial, nous avons utilisé la version 1.6.21 du JDK :
fc.isima.fr/~phan/WebService/jdk-6u21-windows-i586.zip

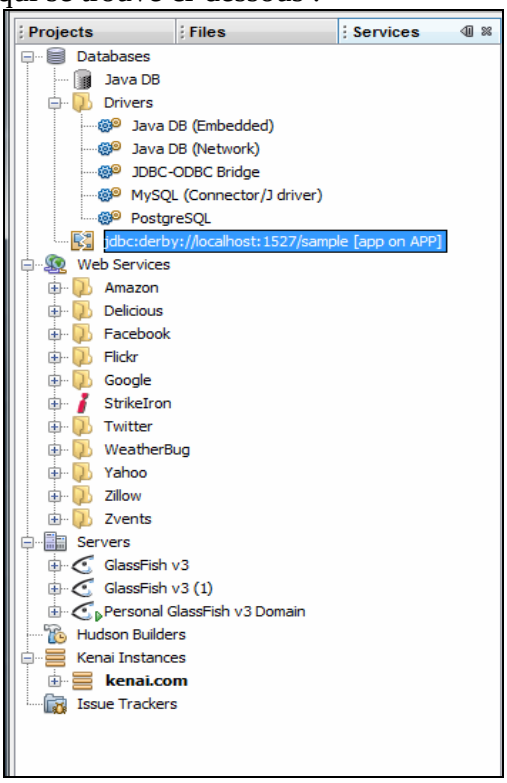
Comme utilitaire de développement, nous allons utiliser NetBeans 6.9. Il sera nécessaire d'avoir au minimum la **version Java** (avec le serveur Glassfish intégré).
Lien pour le téléchargement :
fc.isima.fr/~phan/WebService/netbeans-6.9-ml-java-windows.exe.zip

Suivez le guide d'installation jusqu'à l'endroit où l'on vous demande de choisir un **répertoire d'installation** de **Glassfish**. À ce moment là prenez soin de préciser un **chemin d'accès sans espace**. Pour les utilisateurs de Windows nous conseillons : « C:\glassfish3 » (Ceci permettra d'éviter un bug des versions récentes de Glassfish et Java).

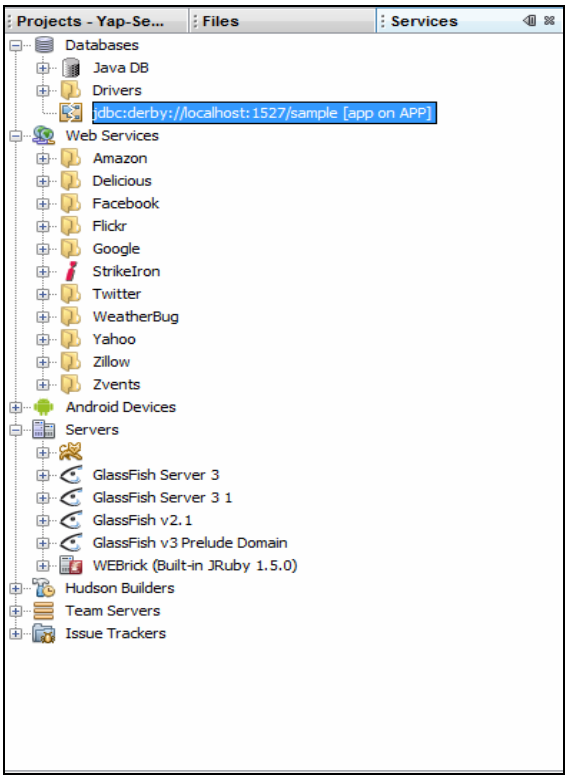


1.7) Configurer Netbeans

Démarrer **NetBeans**. Aller dans **Services**.
En fonction de la machine sur laquelle vous êtes, de la version de Netbeans et de Glassfish ainsi que des éventuelles bases de données déjà installées vous devriez obtenir quelque chose ressemblant à ce qui se trouve ci-dessous :

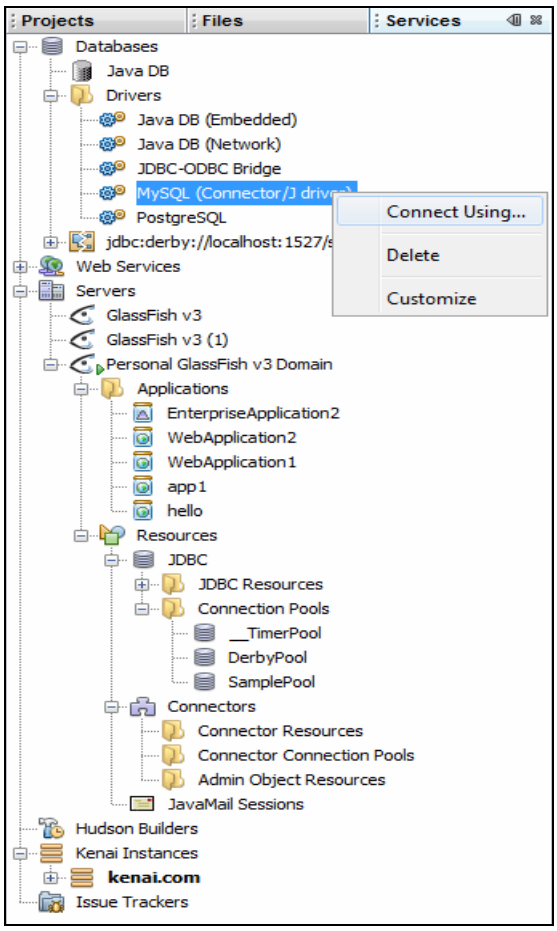


Sous Windows 7, Netbeans 6.8 et Glassfish V3

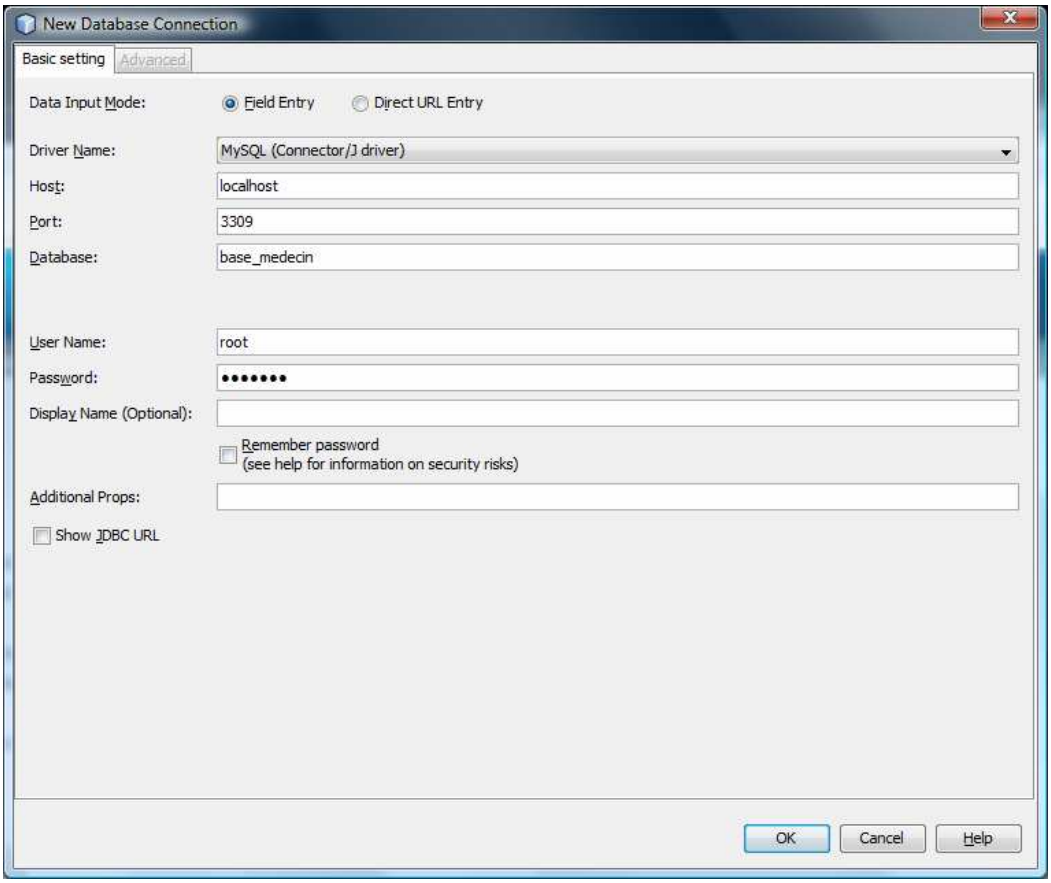


Sous Windows XP, Netbeans 6.9 et Glassfish V3

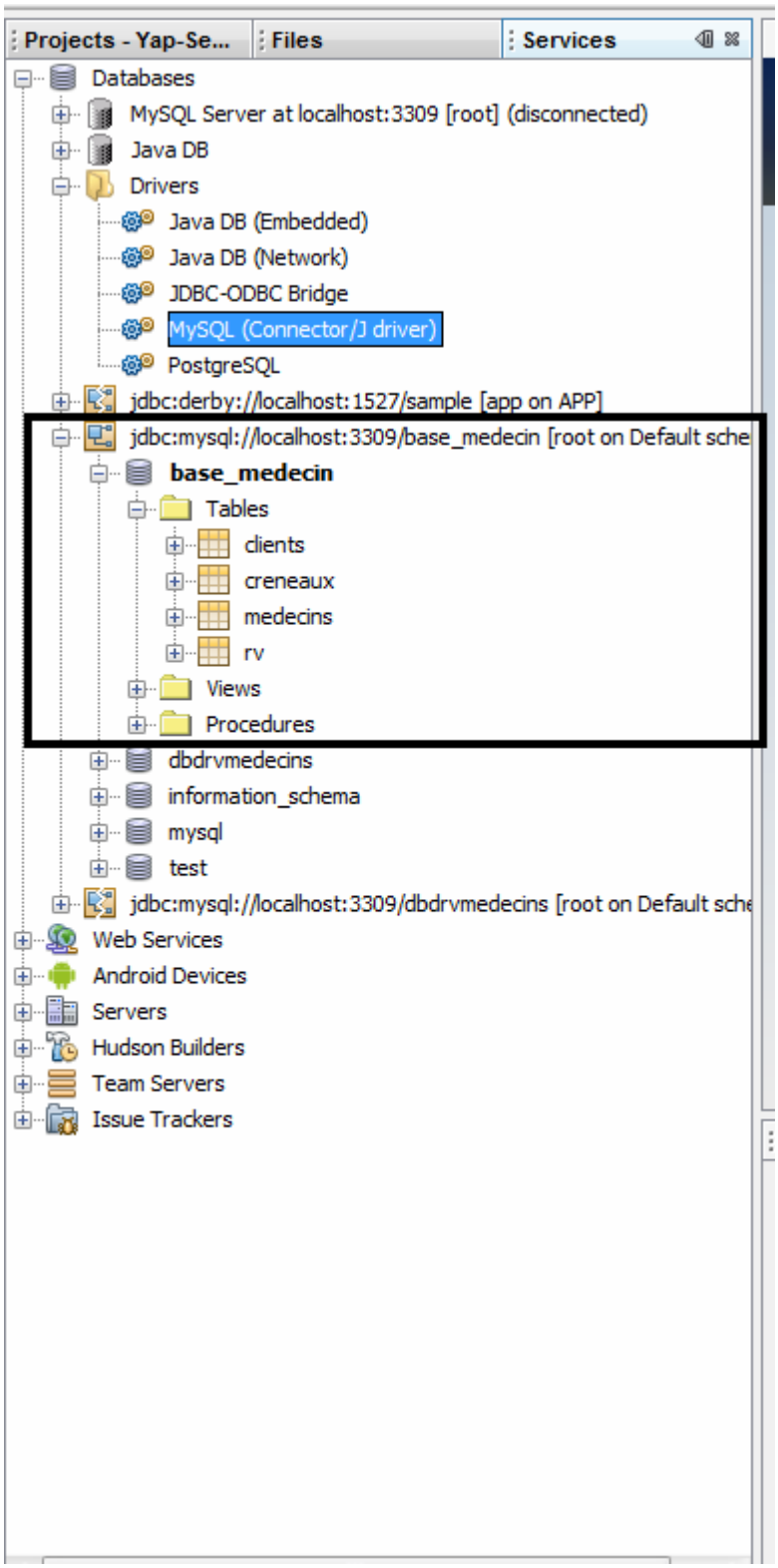
Dans la section Drivers, choisir MySQL et par un clic droit faire **Connect Using**.



Et choisir la base **base_medecin**.

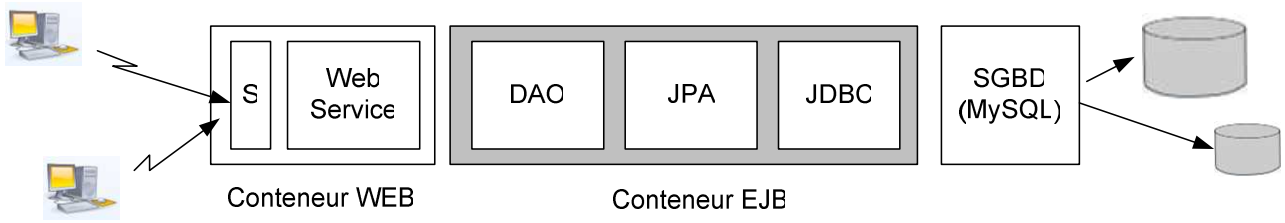


Si la connexion réussie, on obtient alors :



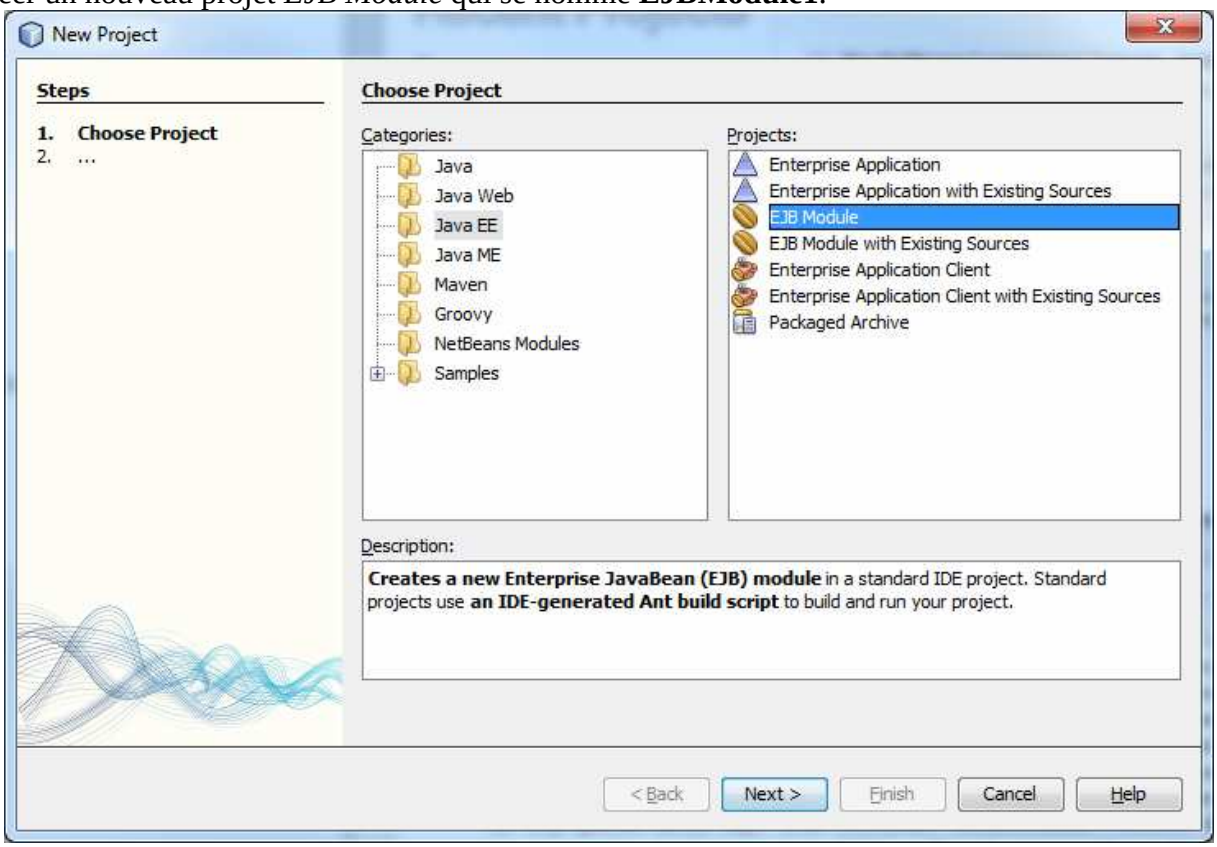
Partie 2. Création d'un EJB

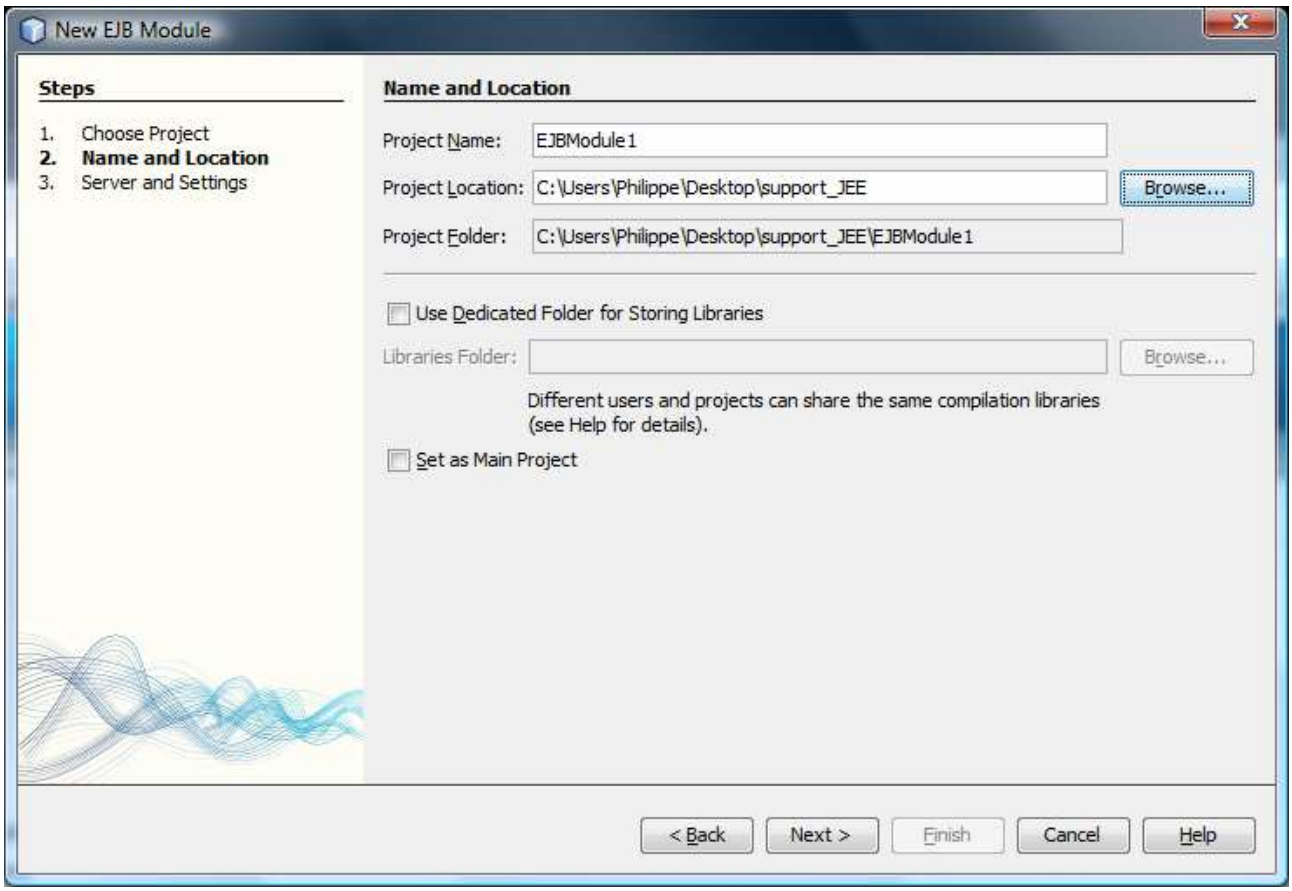
Nous venons de créer la base de données et de configurer la partie SGBD. Nous nous intéressons à la définition du conteneur EJB.

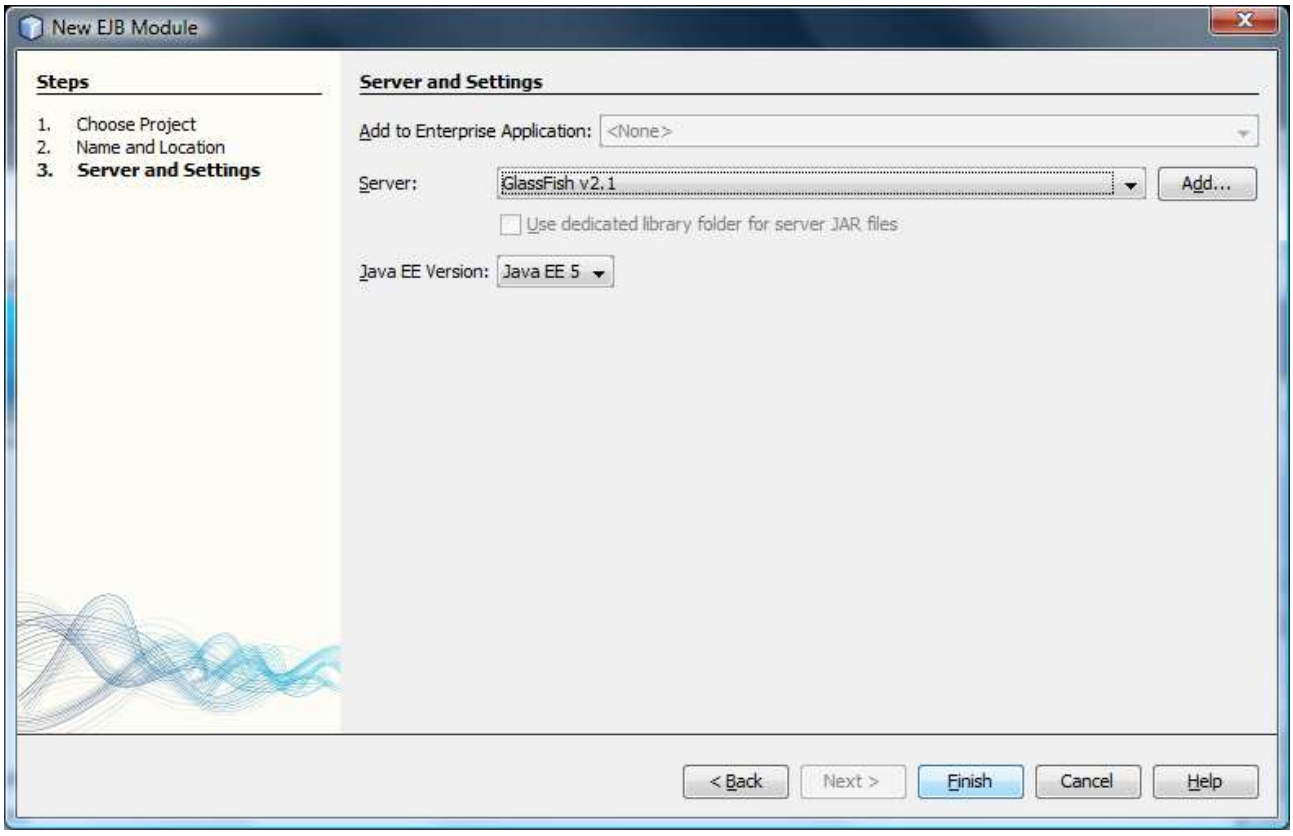


2.1. Création d'un conteneur EJB

Créer un nouveau projet *EJB Module* qui se nomme **EJBModule1**.

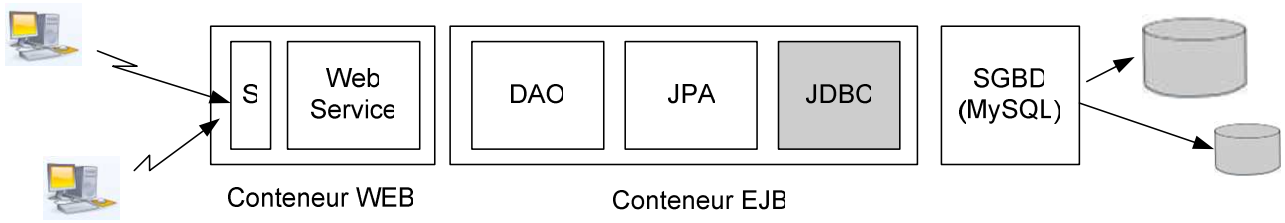


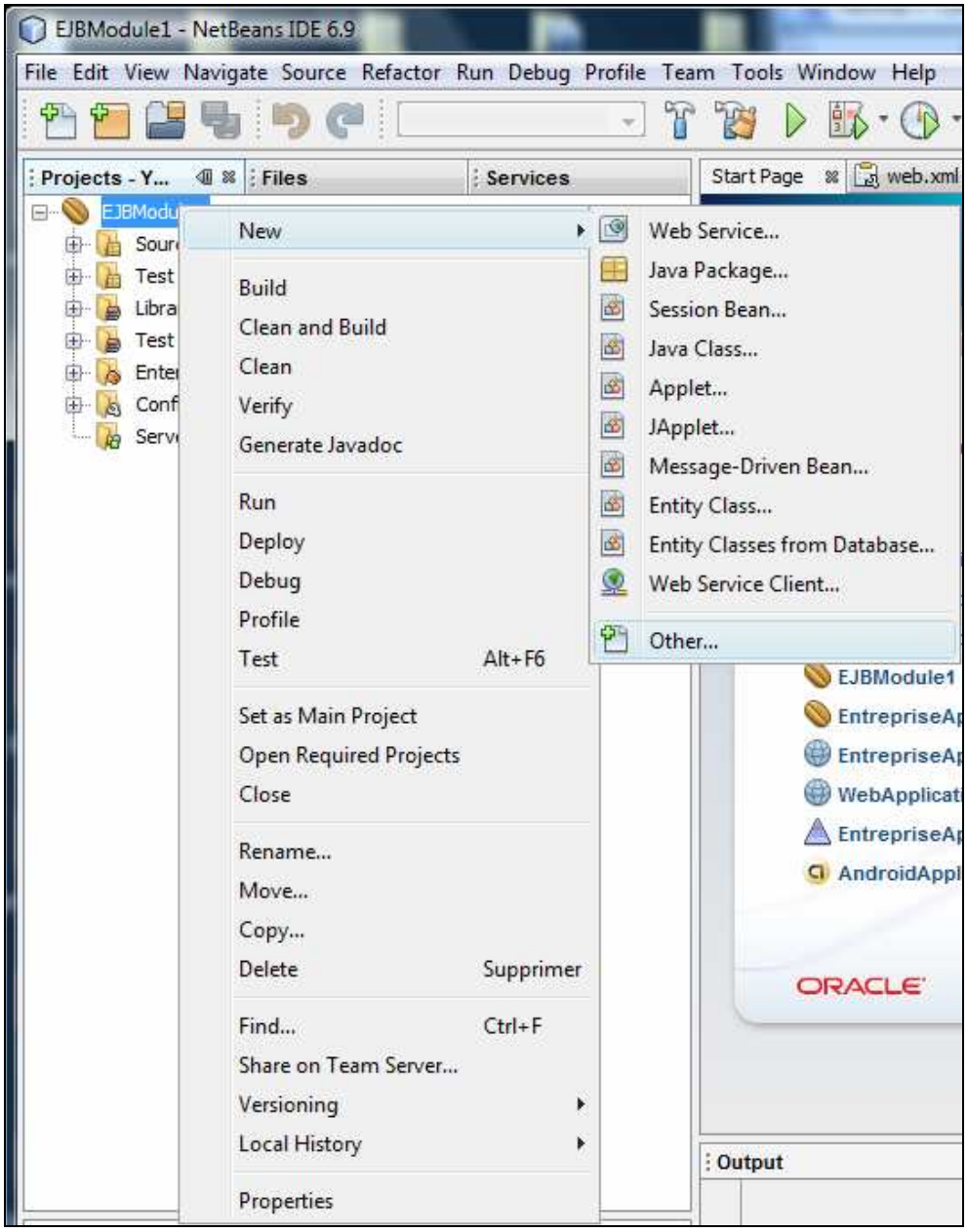

New EJB Module
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
Name and Location
Project Name: EJBModule1
Project Location: C:\Users\Philippe\Desktop\support_JEE
Browse...
Project Folder: C:\Users\Philippe\Desktop\support_JEE\EJBModule1
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries
(see Help for details).
Set as Main Project
< Back
Next >
Finish
Cancel
Help

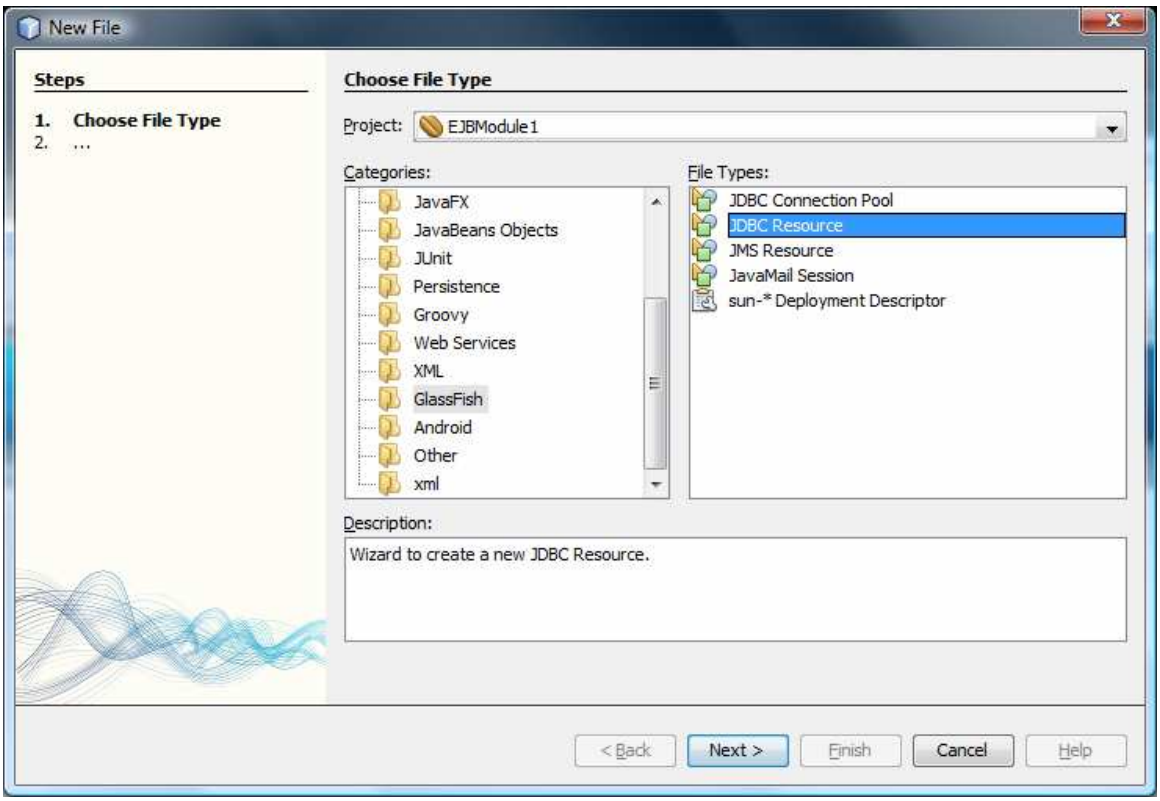

New EJB Module
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
Server and Settings
Add to Enterprise Application: <None>
Server: GlassFish v2.1
Add...
Use dedicated library folder for server JAR files
Java EE Version: Java EE 5
< Back
Next >
Finish
Cancel
Help

2.2. Création d'une ressource JDBC au serveur Glassfish

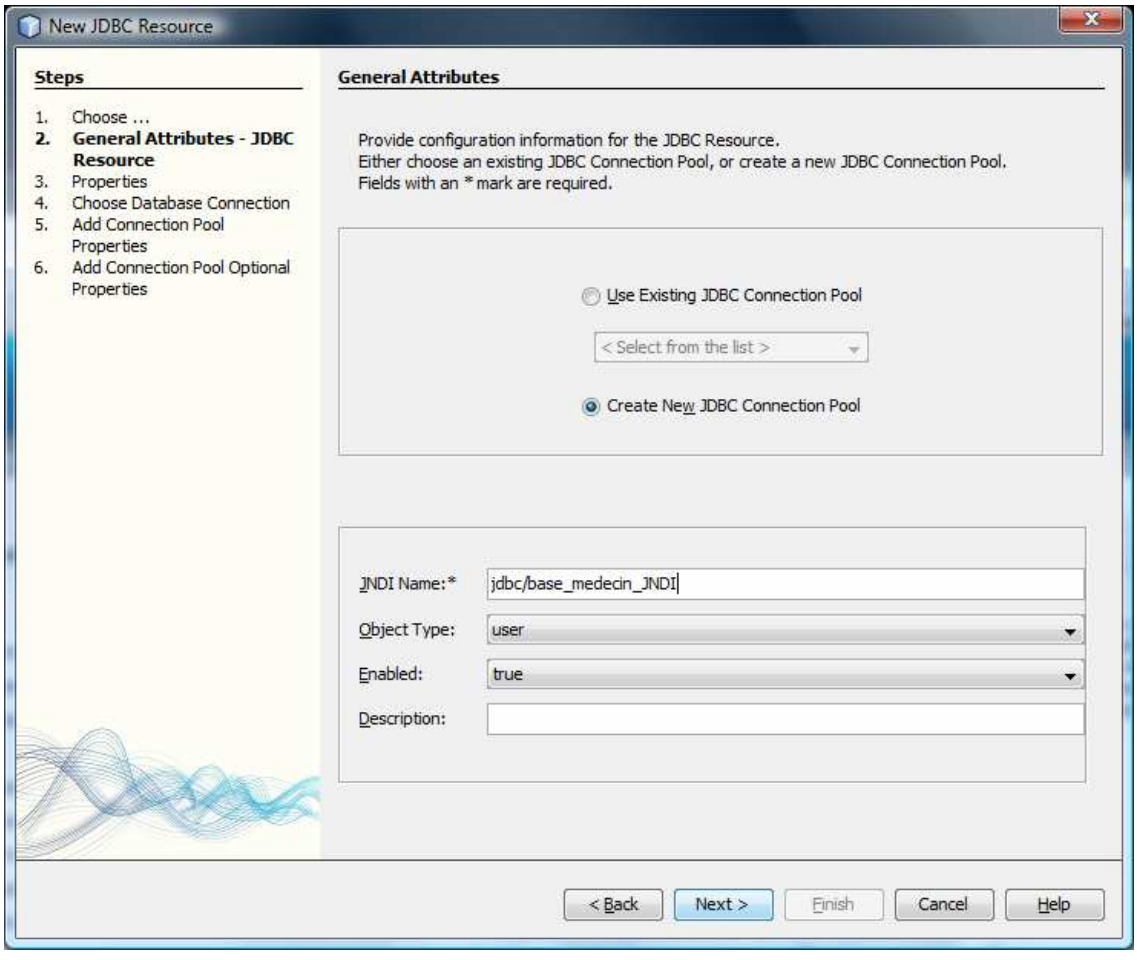
Il s'agit de la première étape.



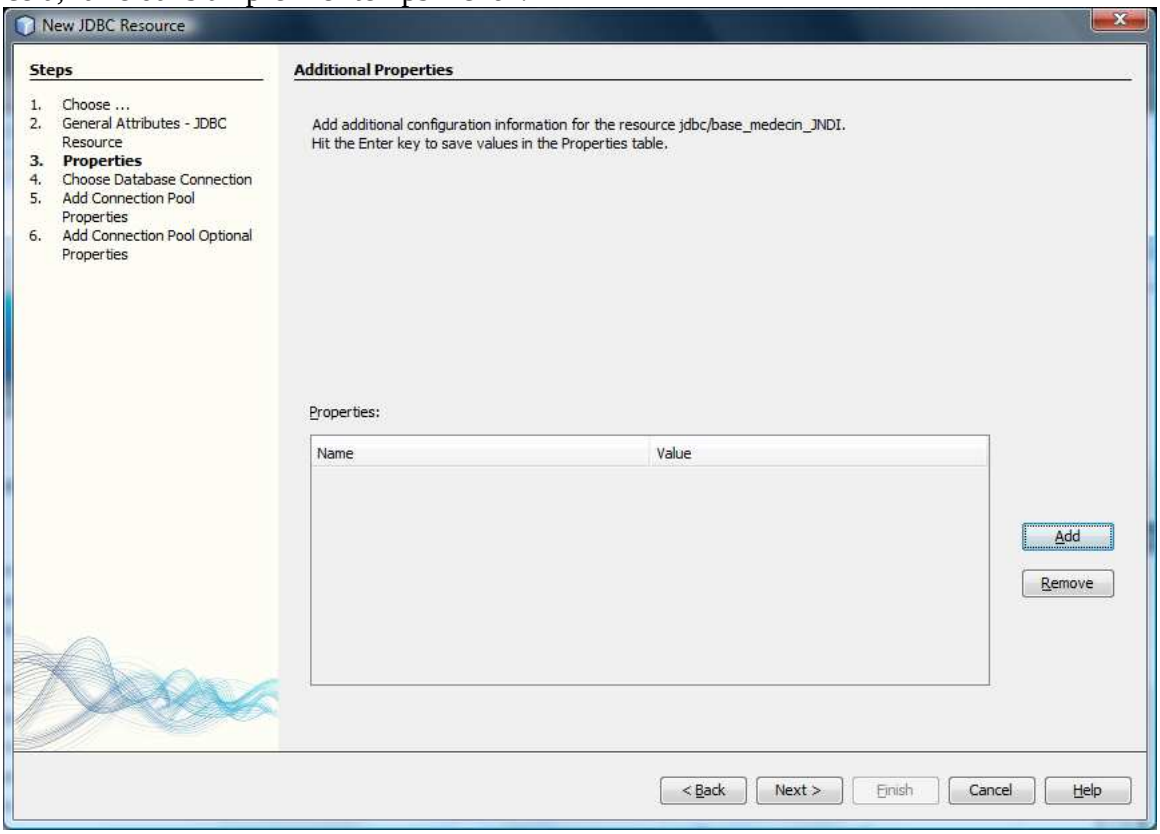




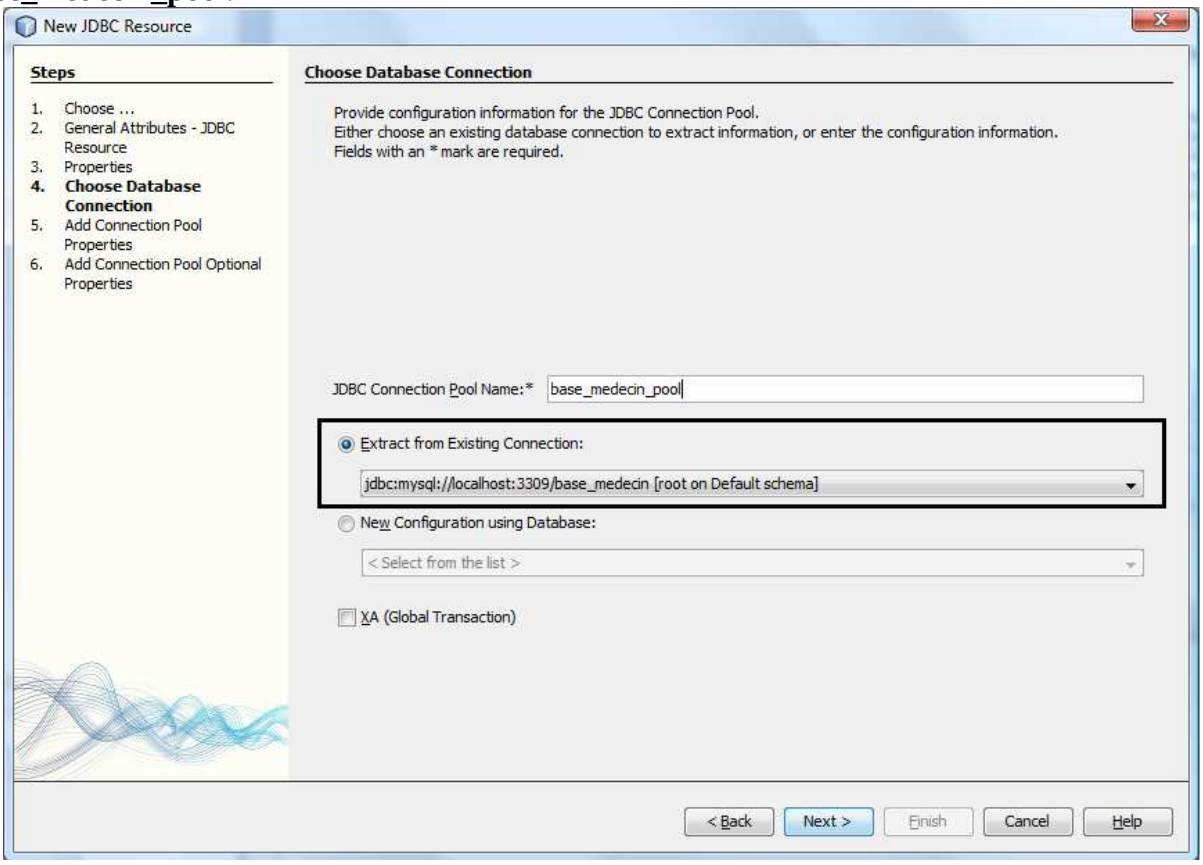
Maintenant nous allons donner un nom JNDI (Java Naming and Directory Interface) à notre ressource. Ce nom sera celui utilisé par le serveur d'application pour « retrouver » la ressource. Afin d'éviter toute confusion, le nom JNDI sera **jdbc/base_medecin_JNDI**:

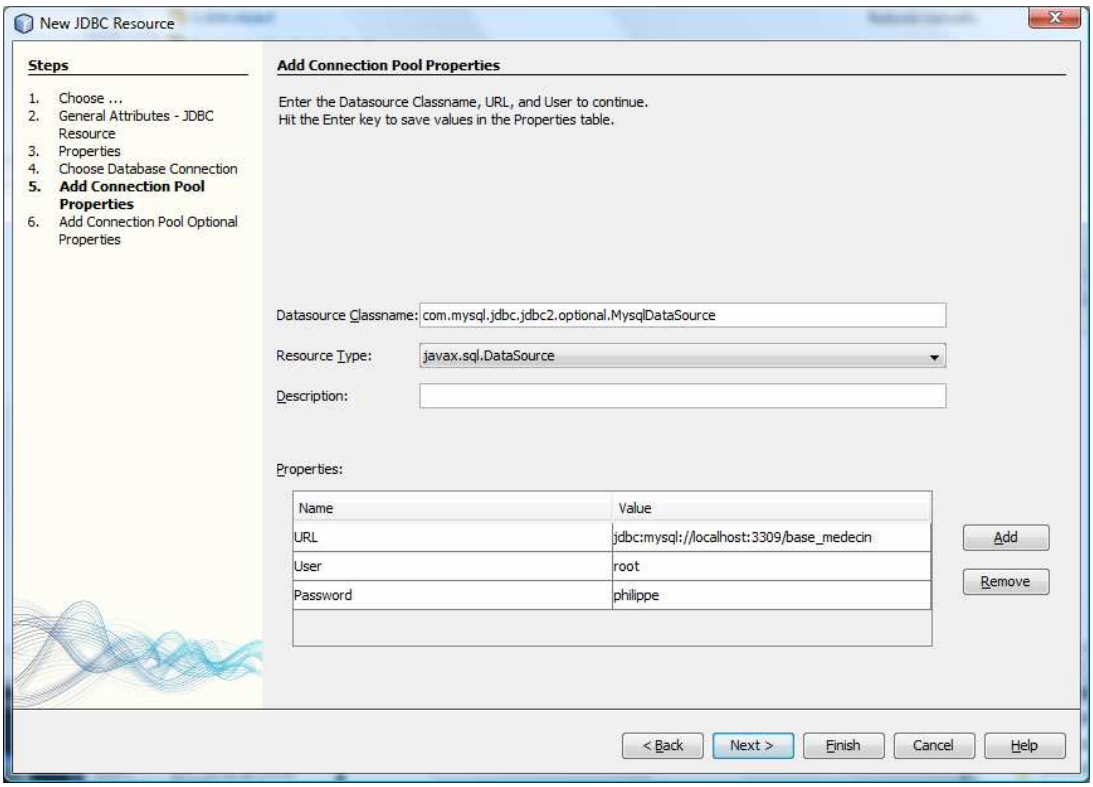


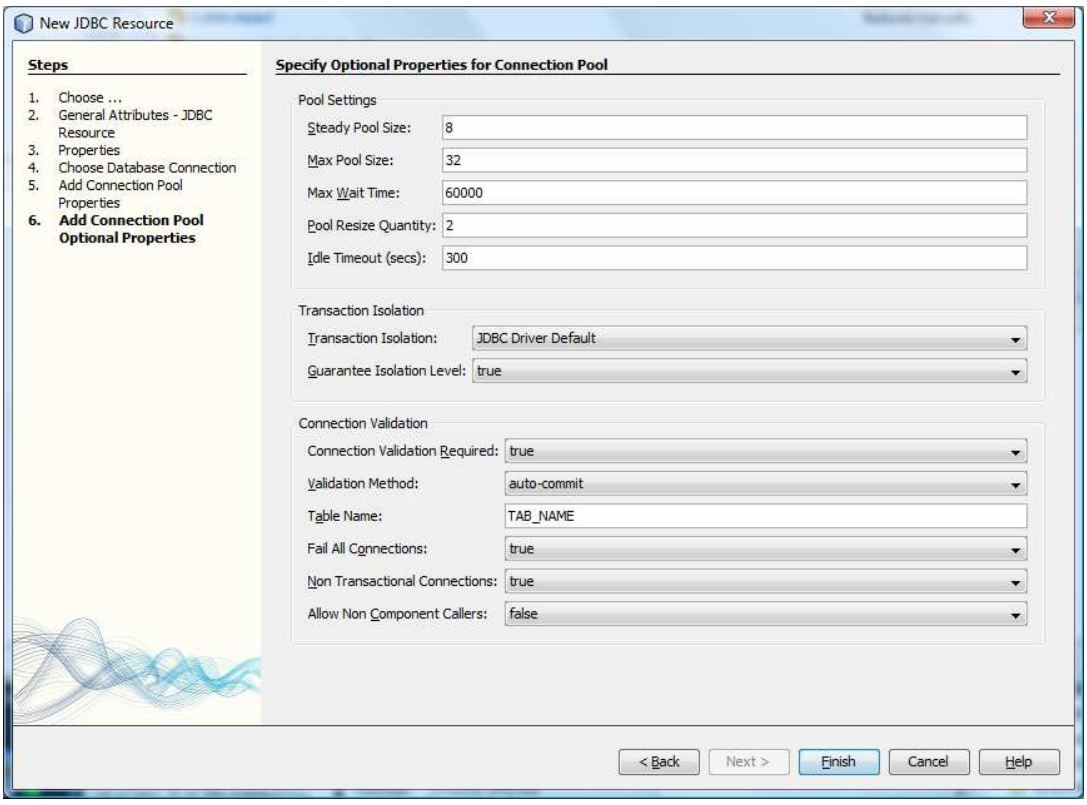
D’autre part, le pool de connexion sera nommé : **base_medecin_pool**.
Pour cela, faire dans un premier temps **Next>**.



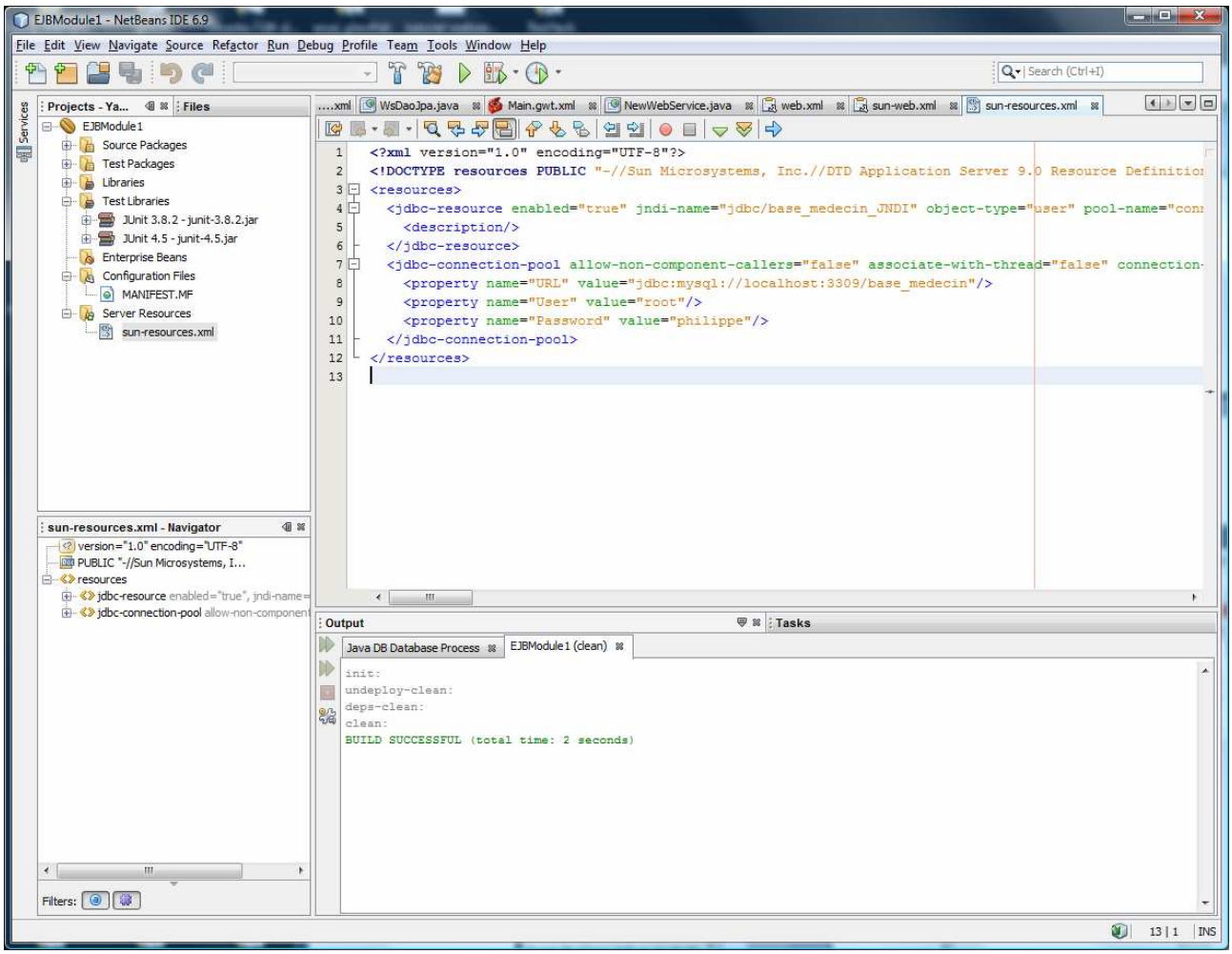
Pensez à choisir la connexion correcte. Ici **base_medecin**. Et donnez lui le nom du **base_medecin_pool**.







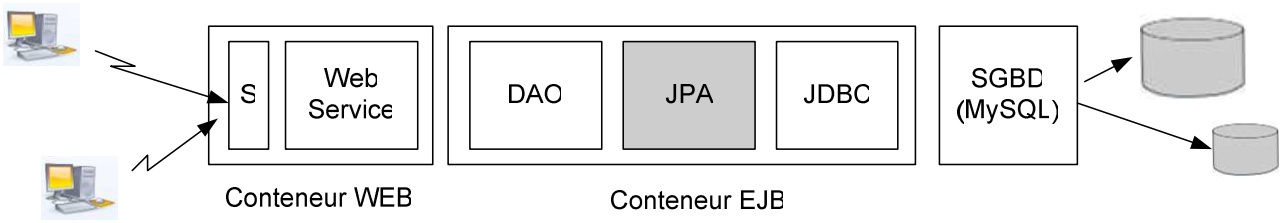
Si on examine le fichier sun-resources.xml, on trouve toute les informations de connexion à la base.




```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 9.0 Resource
Definitions //EN" "http://www.sun.com/software/appserver/dtds/sun-resources_1_3.dtd">
<resources>
  <jdbc-resource enabled="true" jndi-name="jdbc/base_medecin_JNDI" object-type="user" pool-
name="connectionPool">
    <description/>
  </jdbc-resource>
  <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false"
connection-creation-retry-attempts="0" connection-creation-retry-interval-in-seconds="10"
connection-leak-reclaim="false" connection-leak-timeout-in-seconds="0" connection-validation-
method="auto-commit" datasource-classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" fail-all-
connections="false" idle-timeout-in-seconds="300" is-connection-validation-required="false" is-
isolation-level-guaranteed="true" lazy-connection-association="false" lazy-connection-
enlistment="false" match-connections="false" max-connection-usage-count="0" max-pool-size="32" max-
wait-time-in-millis="60000" name="connectionPool" non-transactional-connections="false" pool-resize-
quantity="2" res-type="javax.sql.DataSource" statement-timeout-in-seconds="-1" steady-pool-size="8"
validate-atmost-once-period-in-seconds="0" wrap-jdbc-objects="false">
    <property name="URL" value="jdbc:mysql://localhost:3309/base_medecin"/>
    <property name="User" value="root"/>
    <property name="Password" value="admin"/>
  </jdbc-connection-pool>
</resources>
```

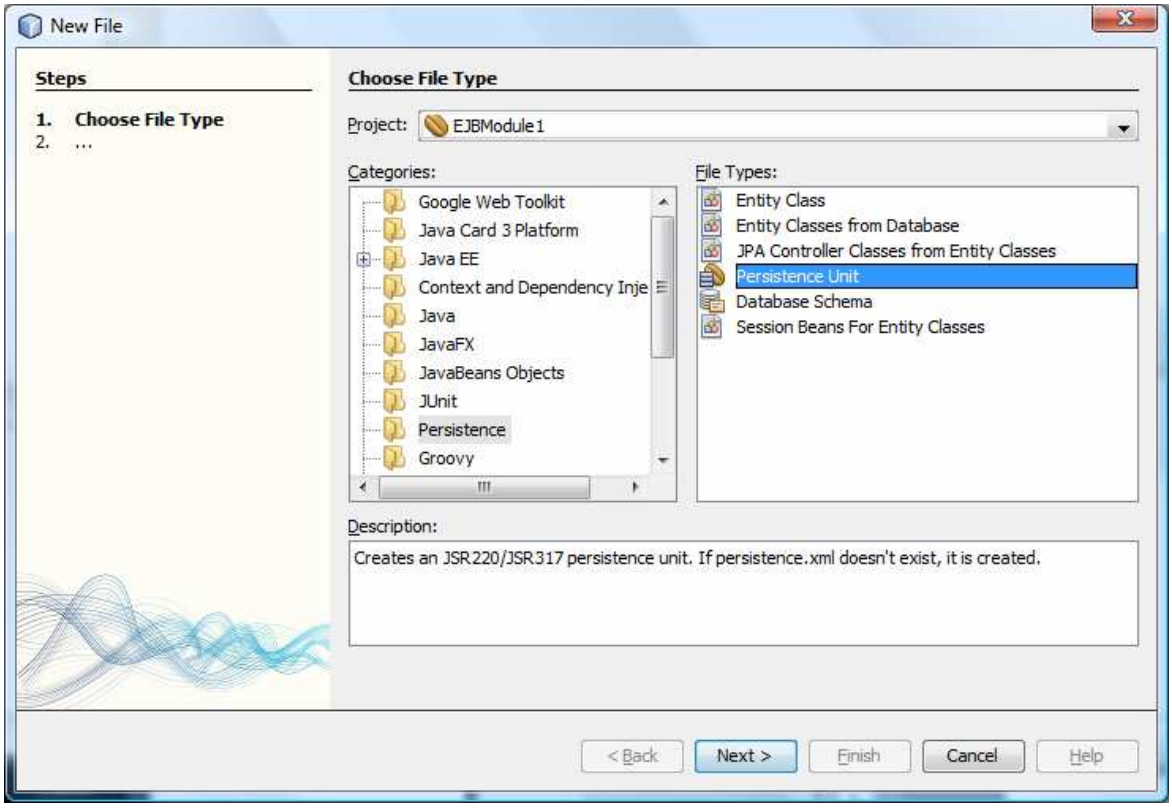
2.3. Création d'une unité de persistance

Il s'agit de la deuxième étape.

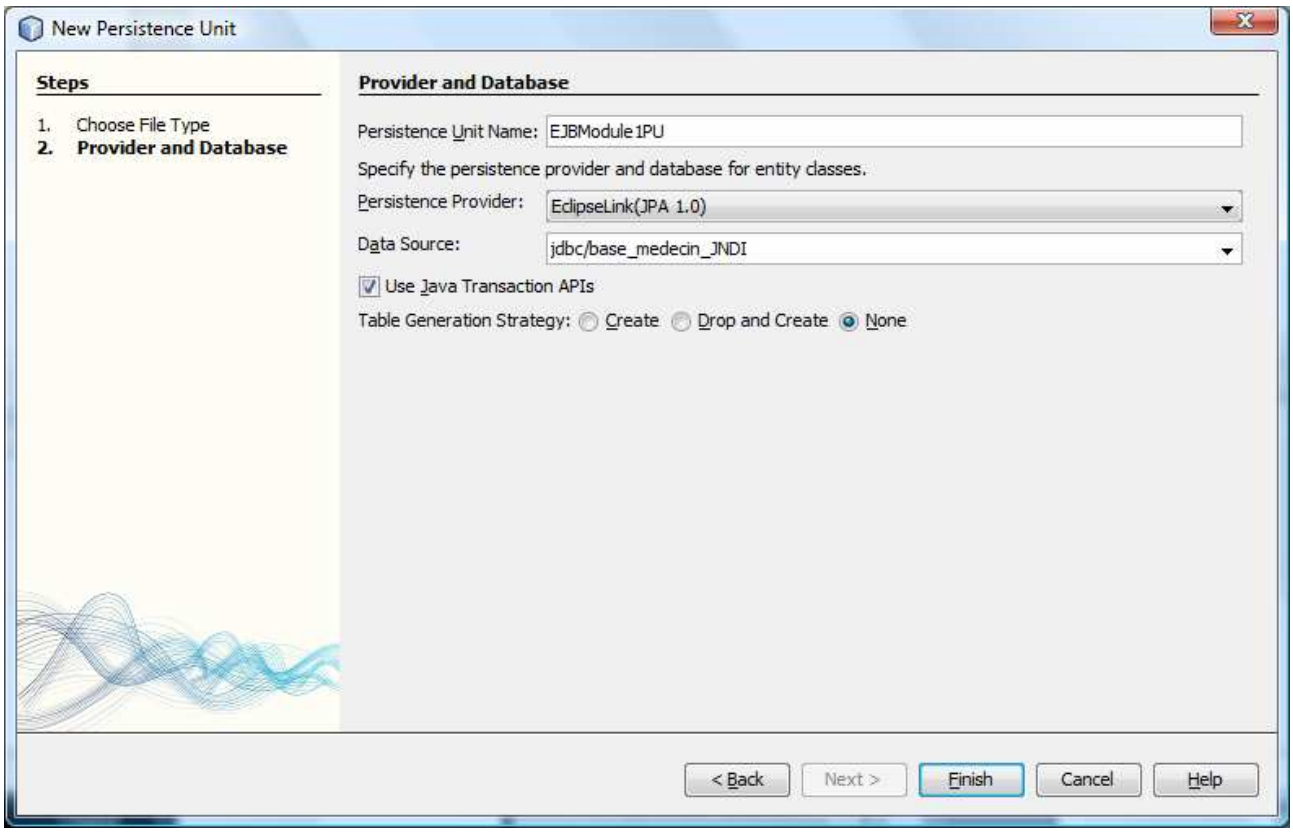


Elle va configurer la couche JPA. Dans notre cas, nous allons utiliser l'implémentation proposée par Eclipse.

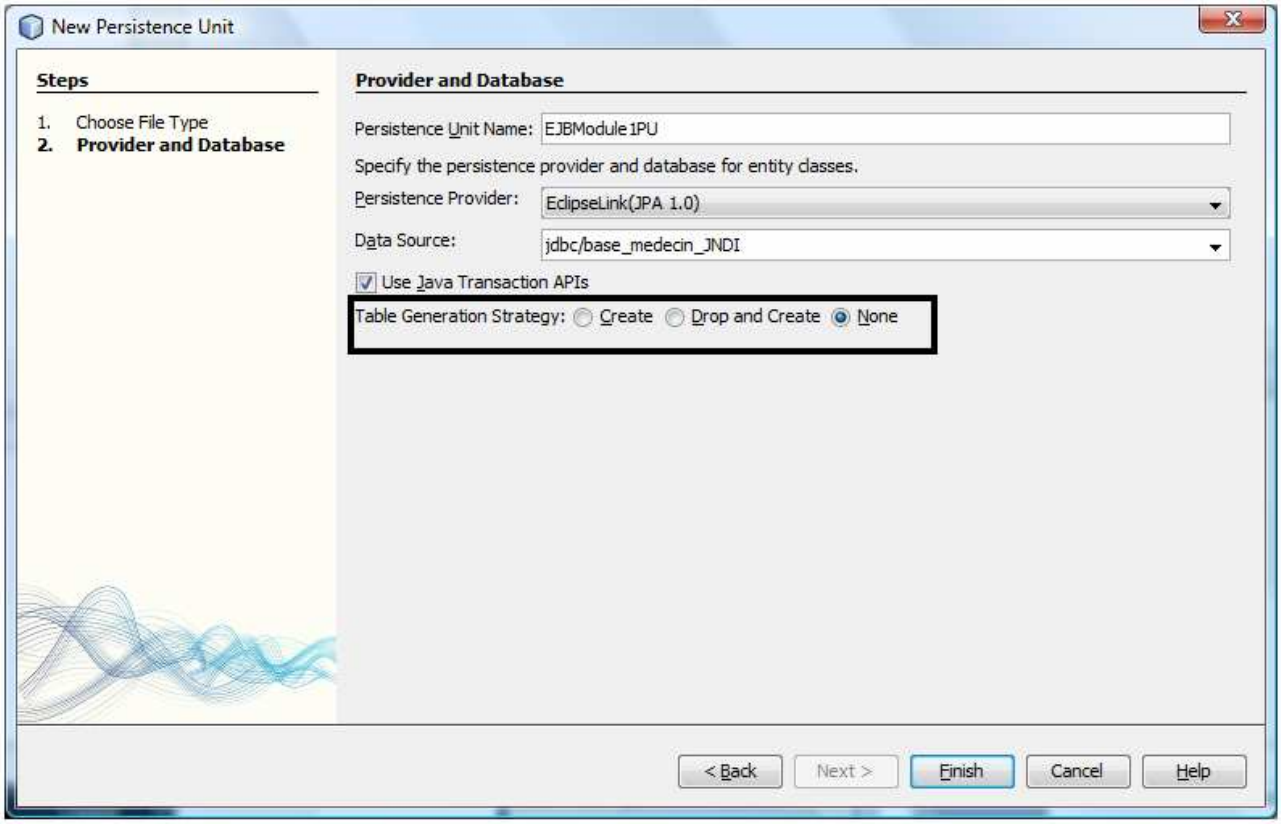
Dans un **premier temps** nous allons créer une unité de persistance (faire Clic Droit sur EJBModule1 et choisir New->Others).



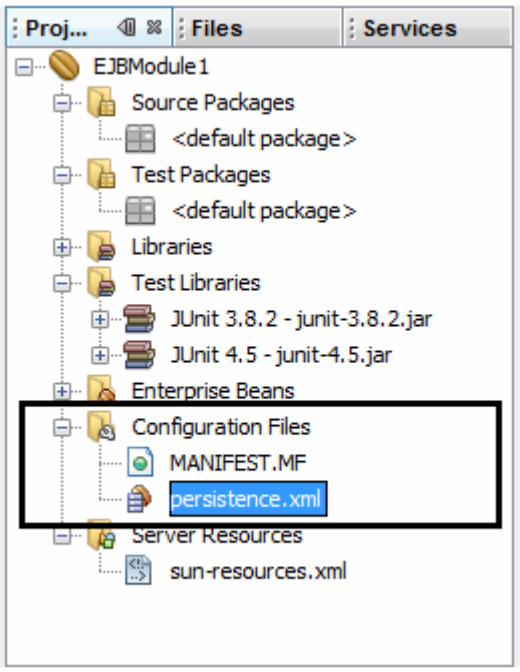
Choisir ensuite comme Data Source : base_medecin_JNDI.
Comme fournisseur de service de persistance EclipseLink.



Attention à ne pas choisir une stratégie englobant la génération des tables. Elles existent déjà grâce au script SQL que nous avons utilisé au départ.

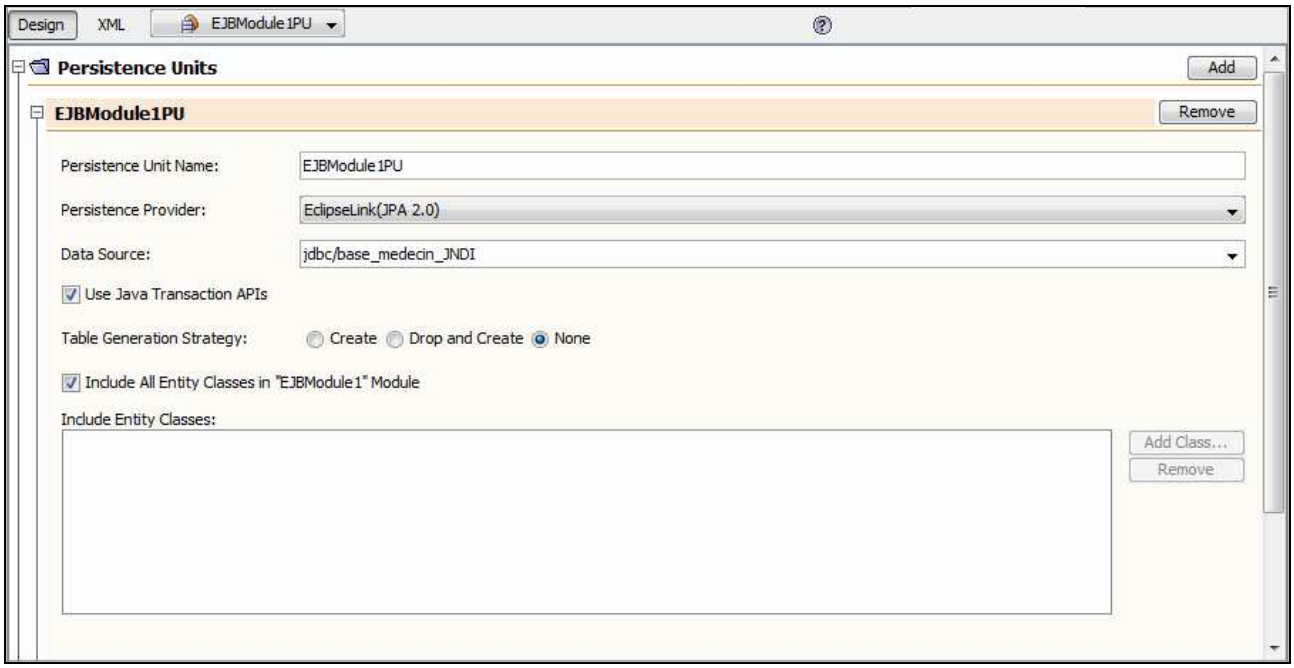


Dans la partie « Configuration Files », le fichier **persistence.xml** apparaît.

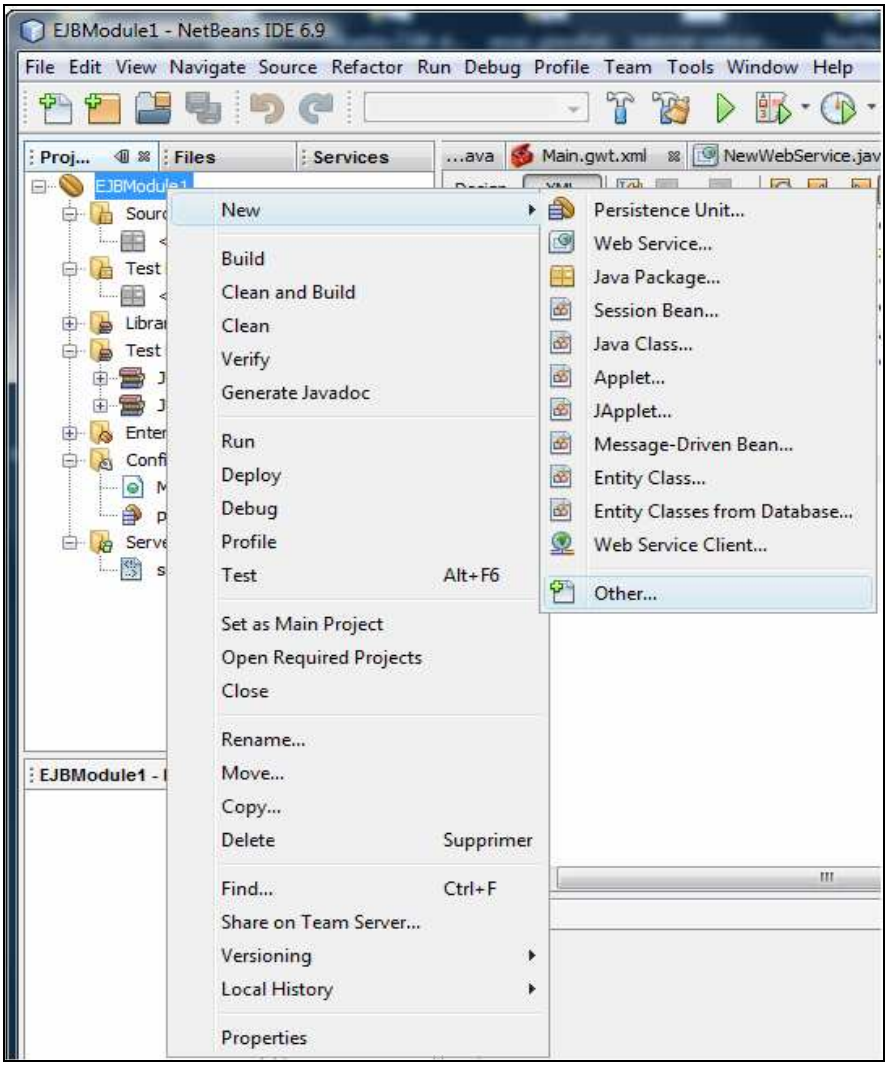


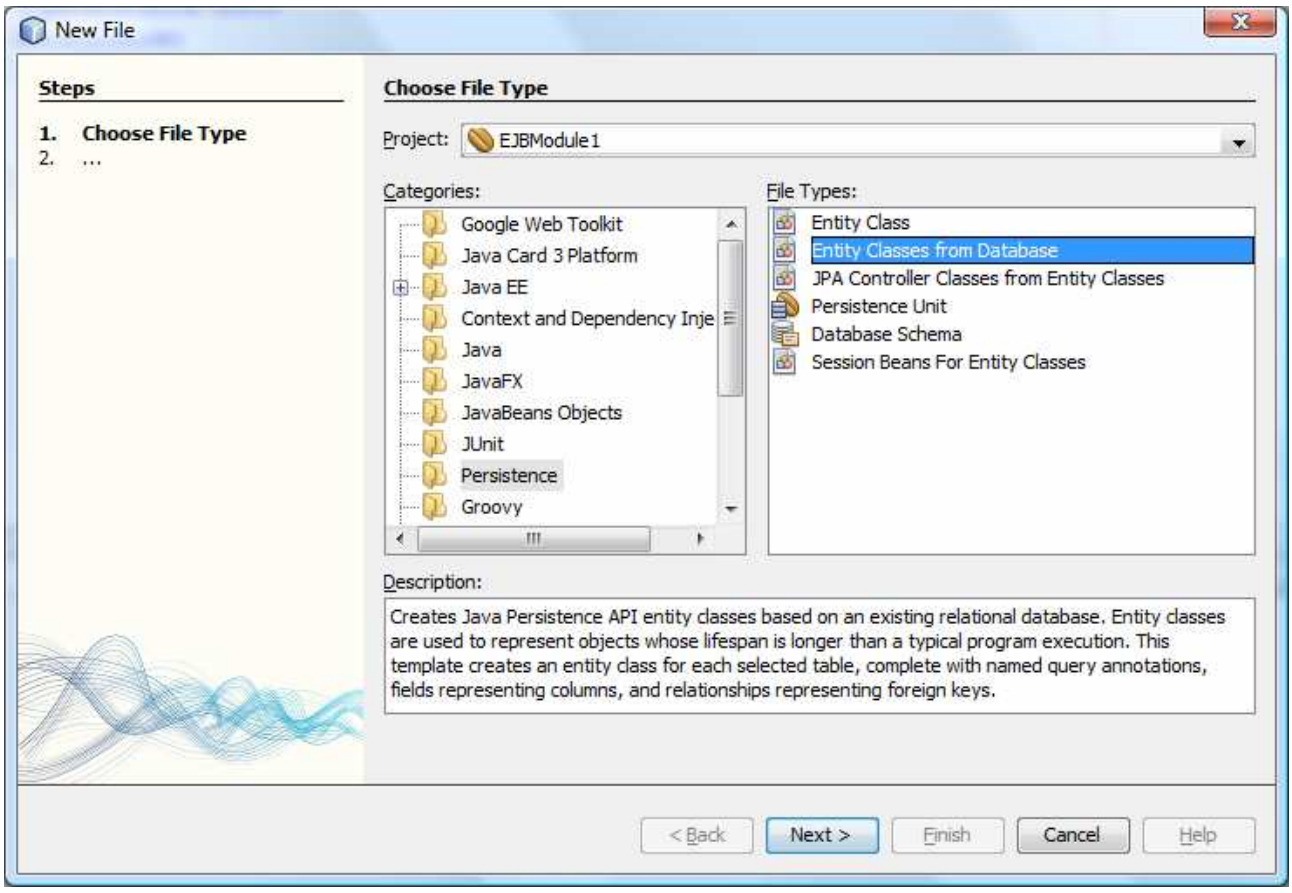

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="EJBModule1PU" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/base_medecin_JNDI</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

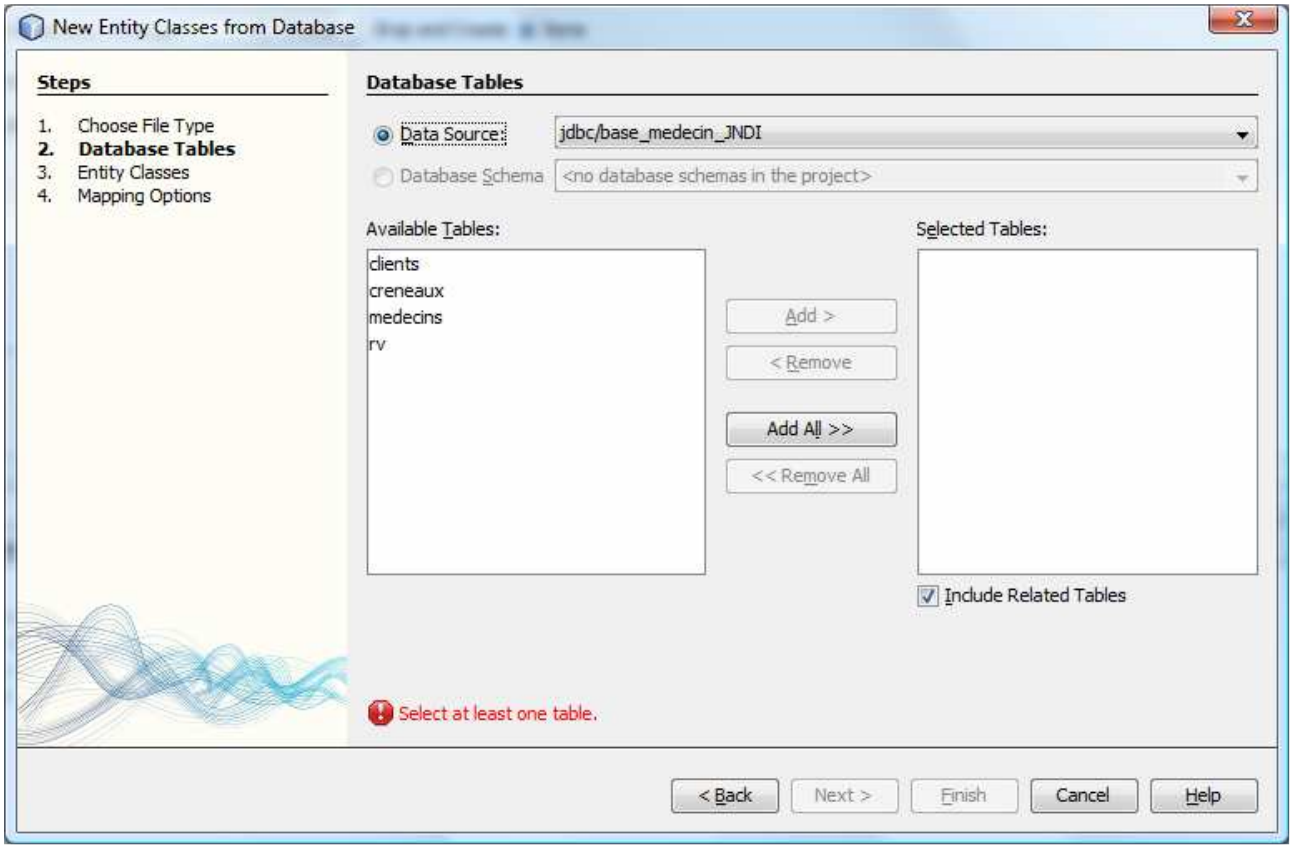
Vérifier que le contenu du fichier persentence.xml est comme suit :

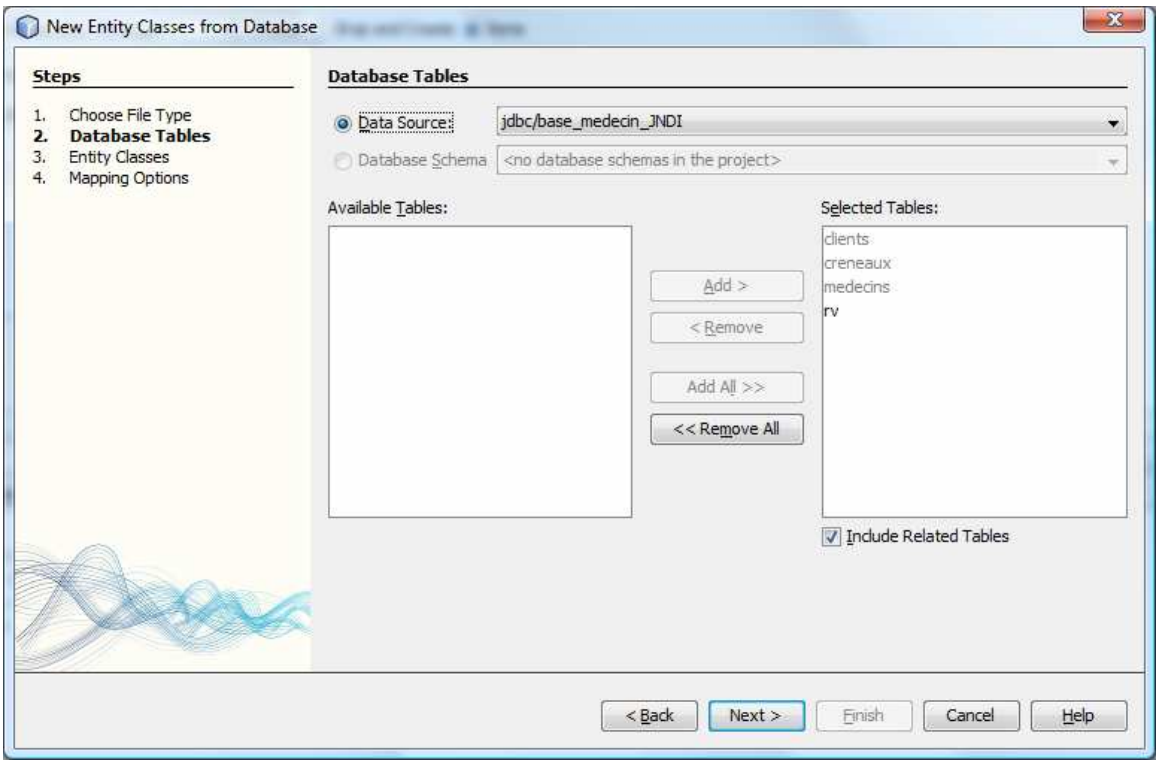


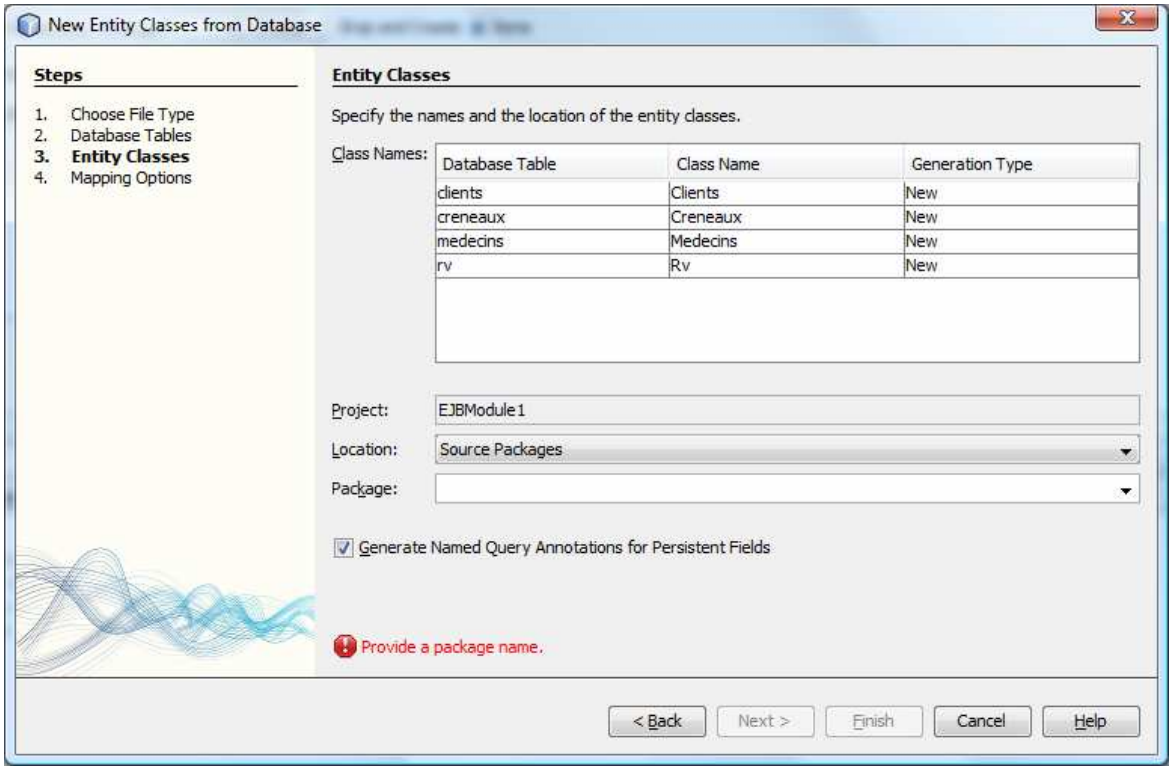
Dans un **deuxième temps**, nous allons créer des entités JPA. Comme précédemment faire Clic Droit / New / Other.



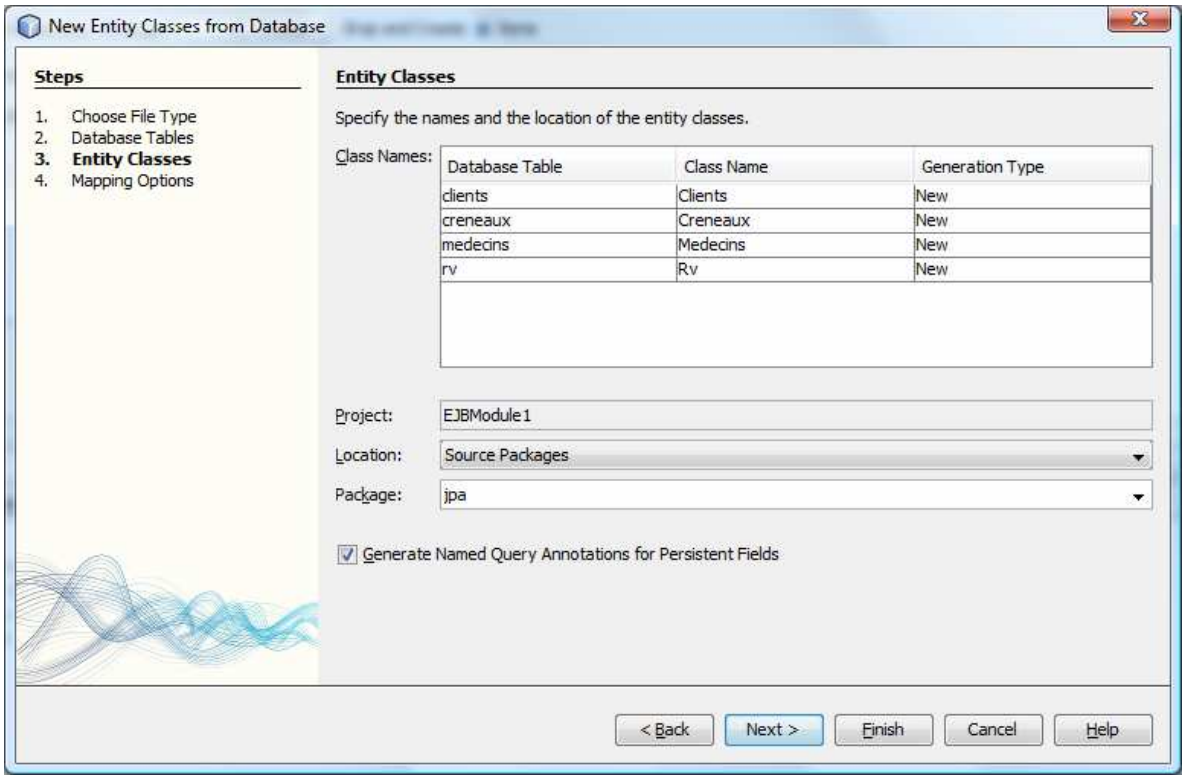

New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project: EJBModule1
Categories:
Google Web Toolkit
Java Card 3 Platform
Java EE
Context and Dependency Inje
Java
JavaFX
JavaBeans Objects
JUnit
Persistence
Groovy
File Types:
Entity Class
Entity Classes from Database
JPA Controller Classes from Entity Classes
Persistence Unit
Database Schema
Session Beans For Entity Classes
Description:
Creates Java Persistence API entity classes based on an existing relational database. Entity classes are used to represent objects whose lifespan is longer than a typical program execution. This template creates an entity class for each selected table, complete with named query annotations, fields representing columns, and relationships representing foreign keys.
< Back
Next >
Finish
Cancel
Help

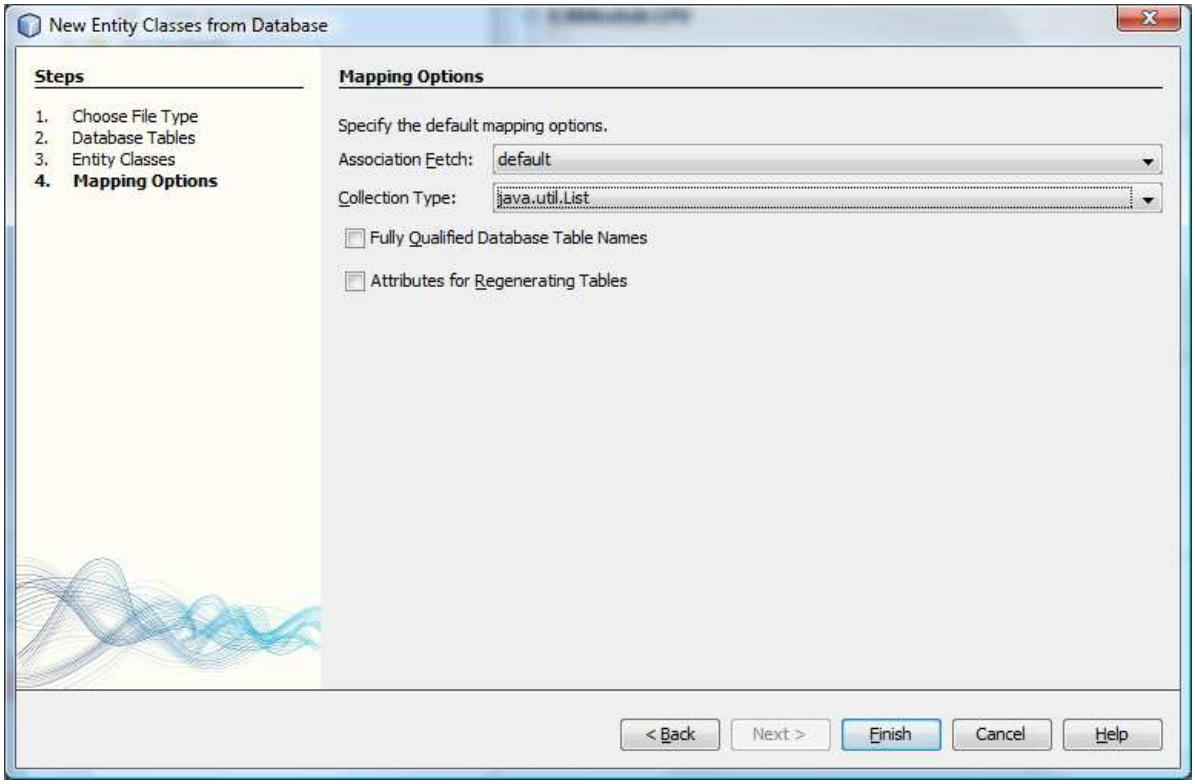

New Entity Classes from Database
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options
Database Tables
Data Source:
jdbc/base_medecin_JNDI
Database Schema
<no database schemas in the project>
Available Tables:
clients
creneaux
medecins
rv
Add >
< Remove
Add All >>
<< Remove All
Selected Tables:
Include Related Tables
Select at least one table.
< Back
Next >
Finish
Cancel
Help


New Entity Classes from Database
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options
Database Tables
Data Source:
jdbc/base_medecin_JNDI
Database Schema
<no database schemas in the project>
Available Tables:
Selected Tables:
clients
creneaux
medecins
rv
Add >
< Remove
Add All >>
<< Remove All
Include Related Tables
< Back
Next >
Finish
Cancel
Help


New Entity Classes from Database
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options
Entity Classes
Specify the names and the location of the entity classes.
Class Names:
Database Table
Class Name
Generation Type
clients
Clients
New
creneaux
Creneaux
New
medecins
Medecins
New
rv
Rv
New
Project:
EJBModule1
Location:
Source Packages
Package:
Generate Named Query Annotations for Persistent Fields
Provide a package name.
< Back
Next >
Finish
Cancel
Help

Etant donné qu'il s'agit de la génération d'entités JPA, on peut choisir **jpa** comme nom de package.





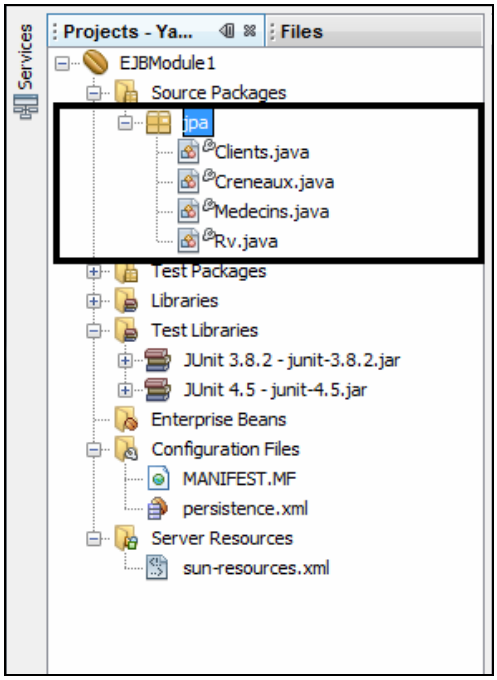
Veuillez à bien choisir « **java.util.List** » pour Collection Type (pour avoir plus de lisibilité).

Rappel :

java.util.Collection : il s'agit de la classe racine de la hiérarchie.

java.util.List : il s'agit d'une extension de collection fournissant les méthodes pour manipuler des listes ordonnées.

Les entités JPA générées apparaissent. Une entité pour chaque table de la base de données



Dans la classe Rv nous ajoutons un nouveau constructeur permettant de créer un rendez vous en donnant une date, un client et un jour.

```
public Rv(Date jour, Clients client, Creneaux creneau)
{
    this.jour = jour ;
    this.clients = client ;
    this.creneaux = creneau ;
}
```

Afin d'éviter d'avoir des références circulaires qui peuvent générer des boucles infinies (notamment lors de la conversion en XML par les web services), il est nécessaire d'ajouter le mot clé « transient » dans les classes **Creneaux**, **Medecins** et **Clients**, à l'endroit où elles « mappent » avec d'autres objets sous forme de listes.

Clients.java

```
54      @OneToMany(cascade = CascadeType.ALL, mappedBy = "idClient")
55      private transient List<Rv> rvList;
56
```

Medecins.java

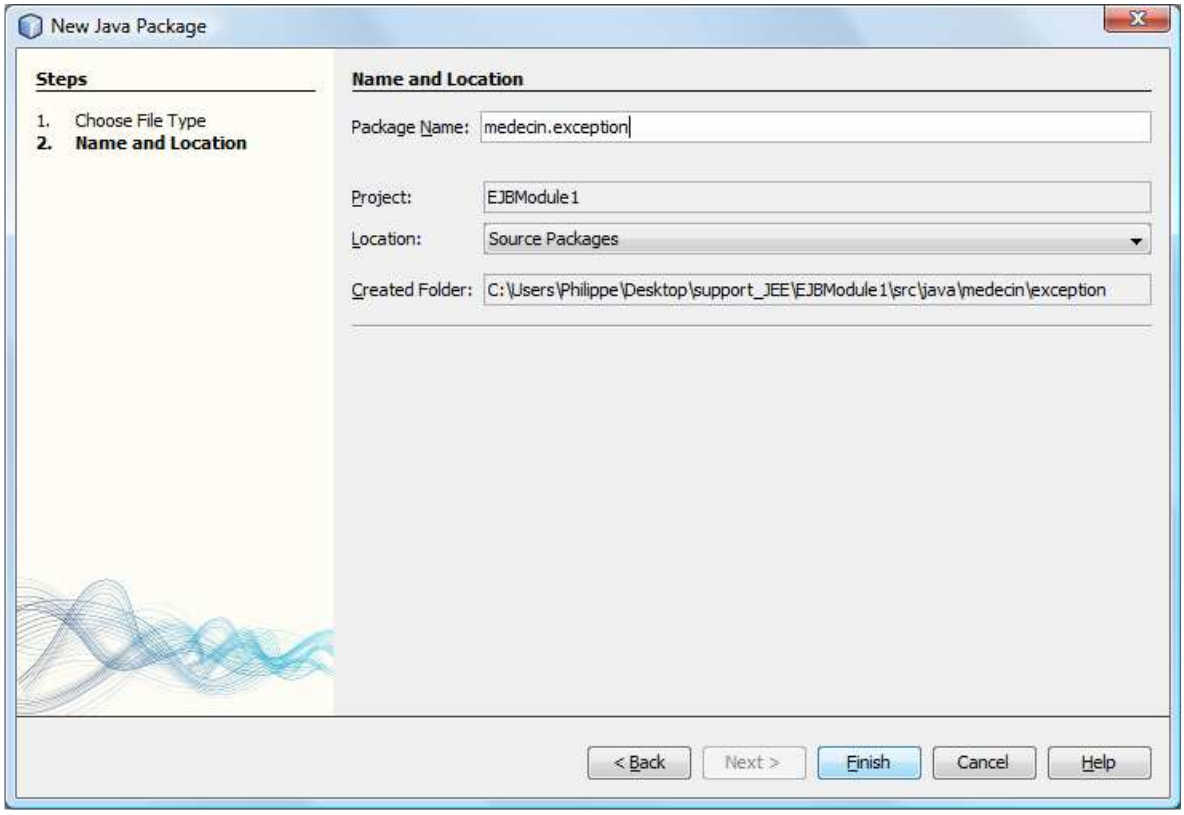
```
54      @OneToMany(cascade = CascadeType.ALL, mappedBy = "idMedecin")
55      private transient List<Creneaux> creneauxList;
56
```

Creneaux.java

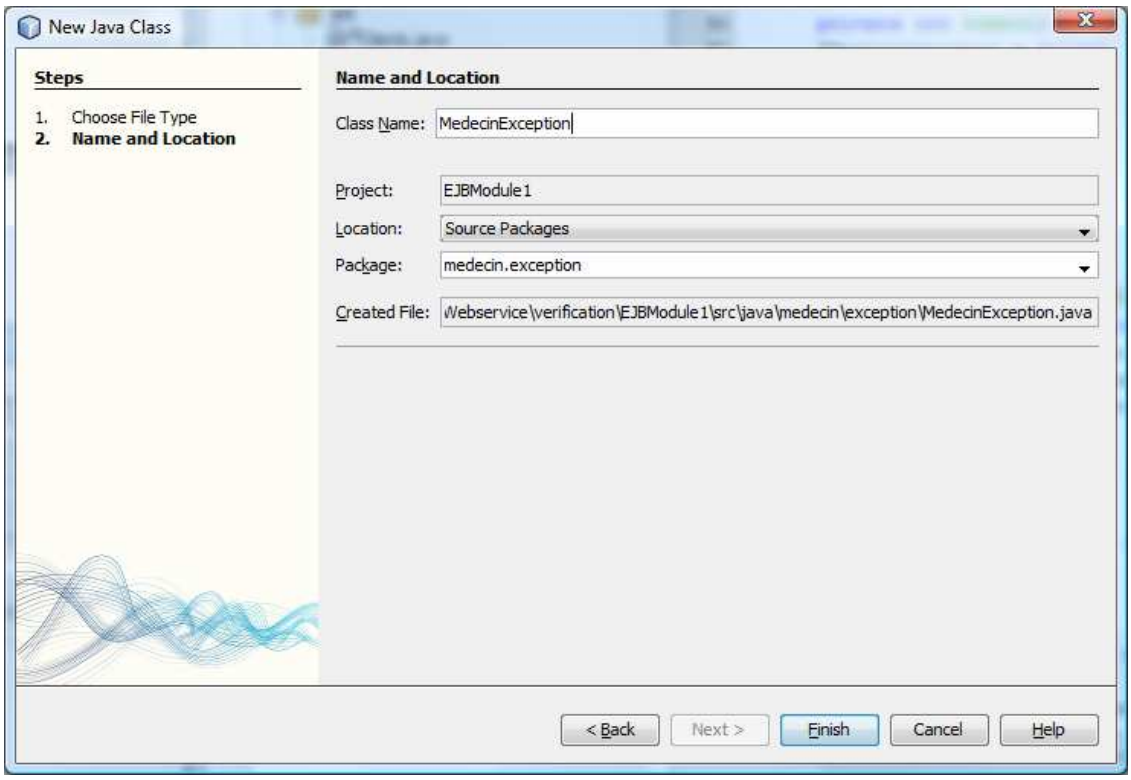
```
63      @OneToMany(cascade = CascadeType.ALL, mappedBy = "idCreneau")
64      private transient List<Rv> rvList;
```

2.4. Création d'une classe spéciale de traitement des exceptions

Créer un nouveau package nommé par exemple **medecin.exception**.



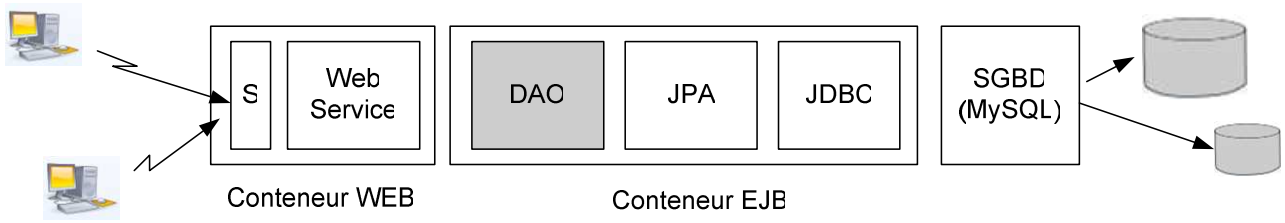
Dans ce package, créer ensuite une classe java nommé par exemple : **MedecinException**.



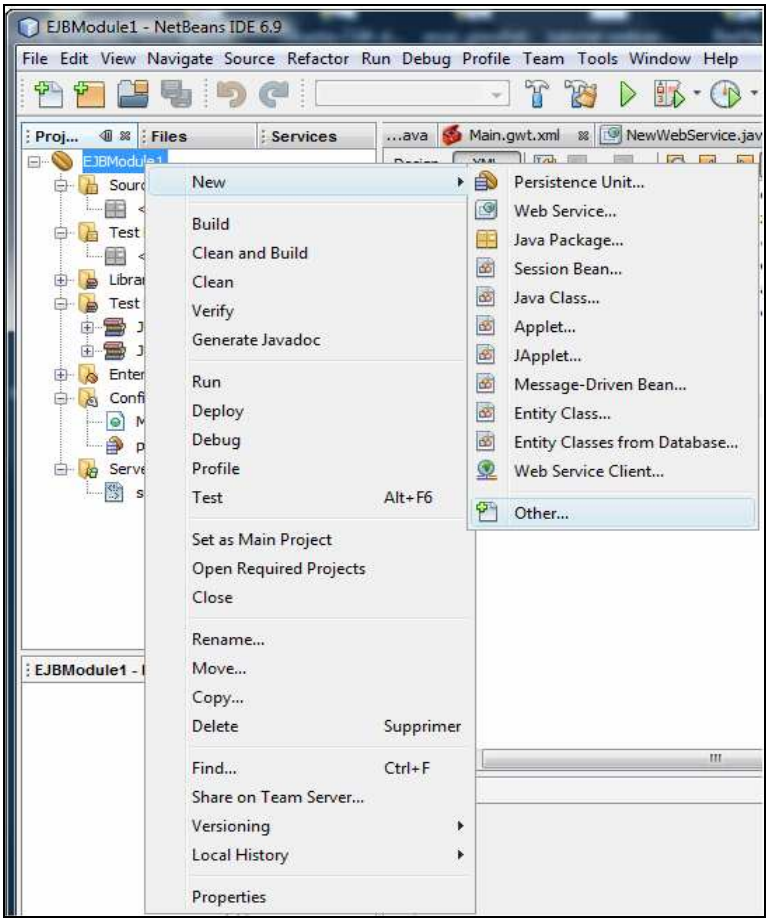
Le code de cette classe pourrait être celui-ci :

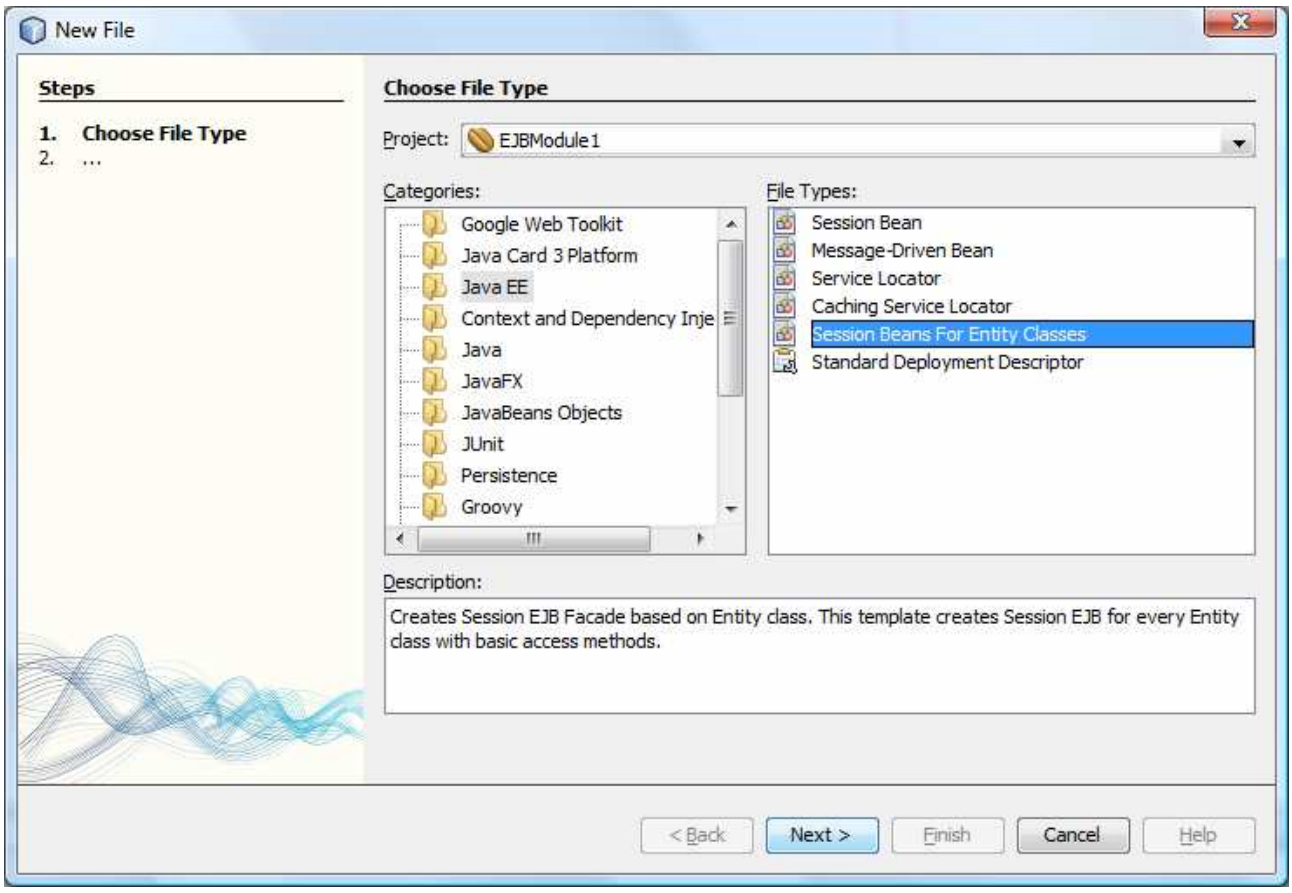
```
package medecin.exception;

import javax.ejb.ApplicationException;

@ApplicationException(rollback=true)

public class MedecinException extends RuntimeException {

  // champs privés
   private int code = 0;
  // constructeurs
public MedecinException() {
   super();
  }
public MedecinException(String message) {
   super(message);
  }
public MedecinException(String message, Throwable cause) {
    super(message, cause);
  }
public MedecinException(Throwable cause) {
  super(cause);
 }
public MedecinException(String message, int code) {
   super(message);
   setCode(code);
 }
public MedecinException(Throwable cause, int code) {
   super(cause);
   setCode(code);
 }
public MedecinException(String message, Throwable cause, int code) {
   super(message, cause);
   setCode(code);
 }
// getters and setters
  public int getCode() {
      return code;
  }
  public void setCode(int code) {
      this.code = code;
  }
}
```

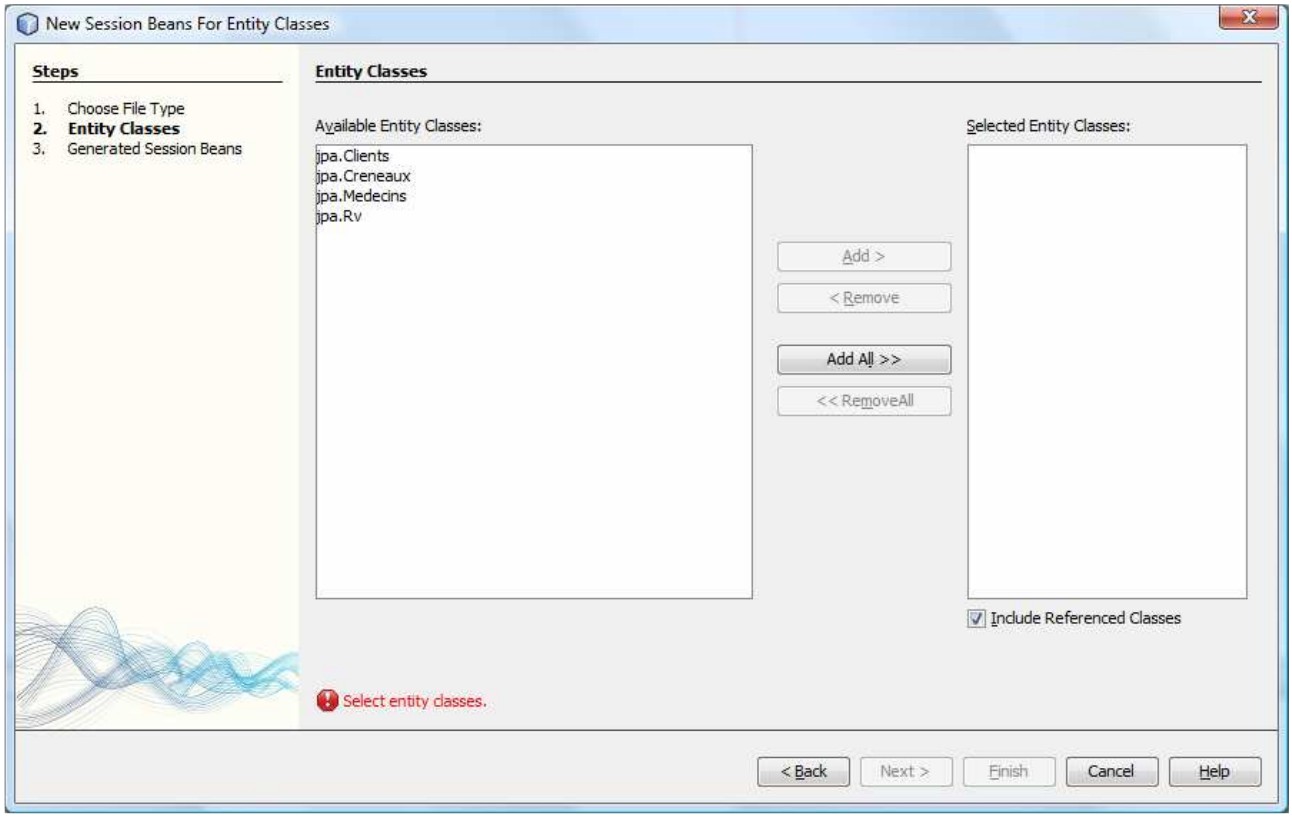
2.5. Création de la couche d'accès aux entités JPA : Beans Session

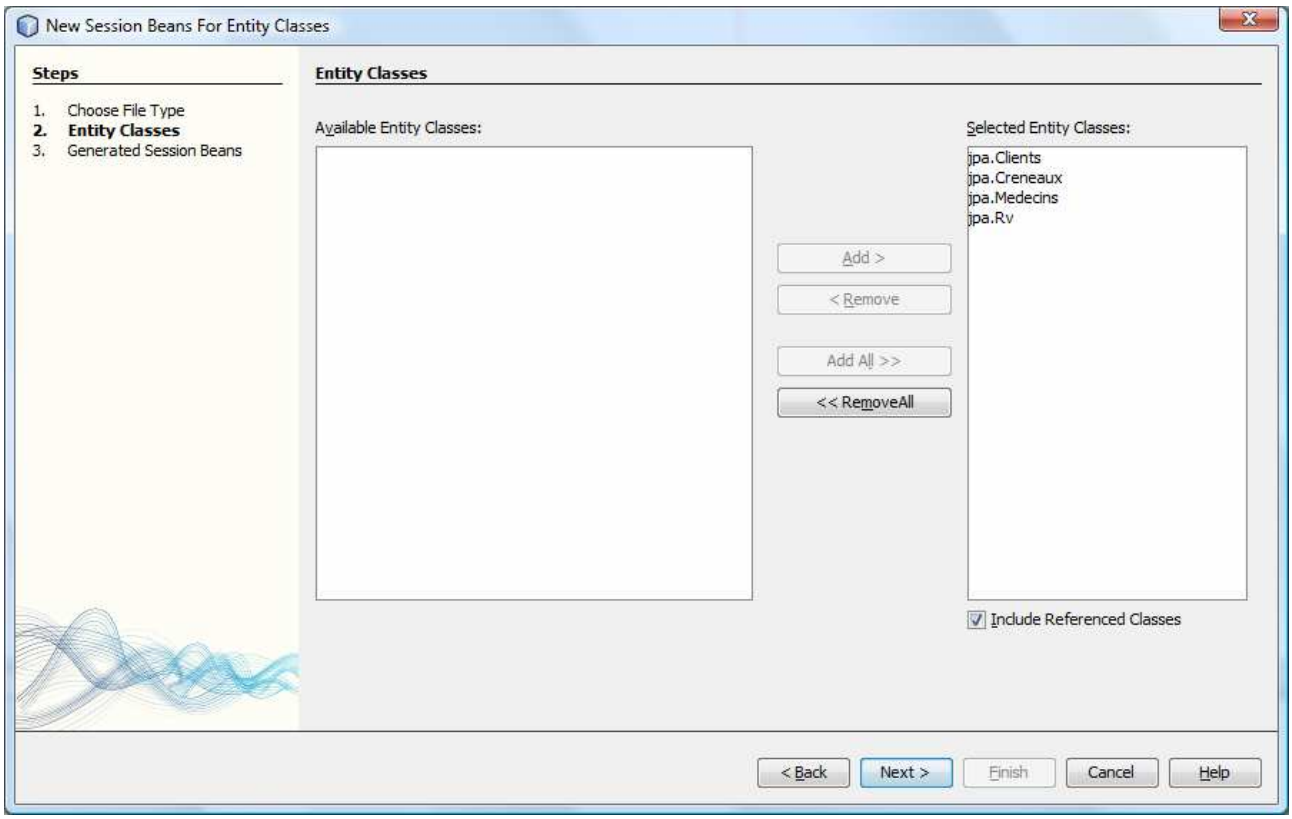


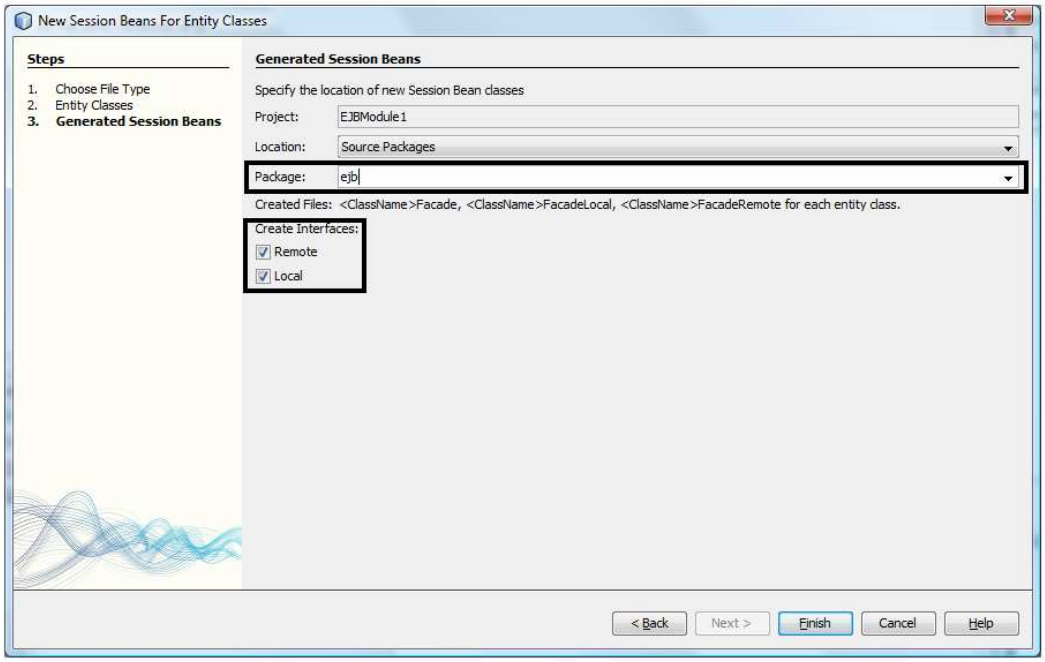
Elle va utiliser la couche JPA. Dans notre cas, nous allons utiliser l'implémentation proposée par EclipseLink.

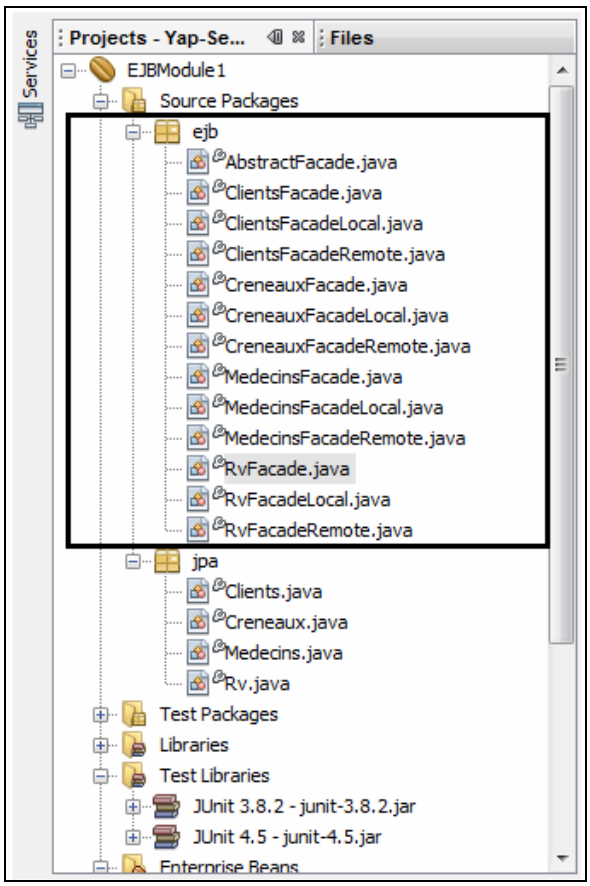


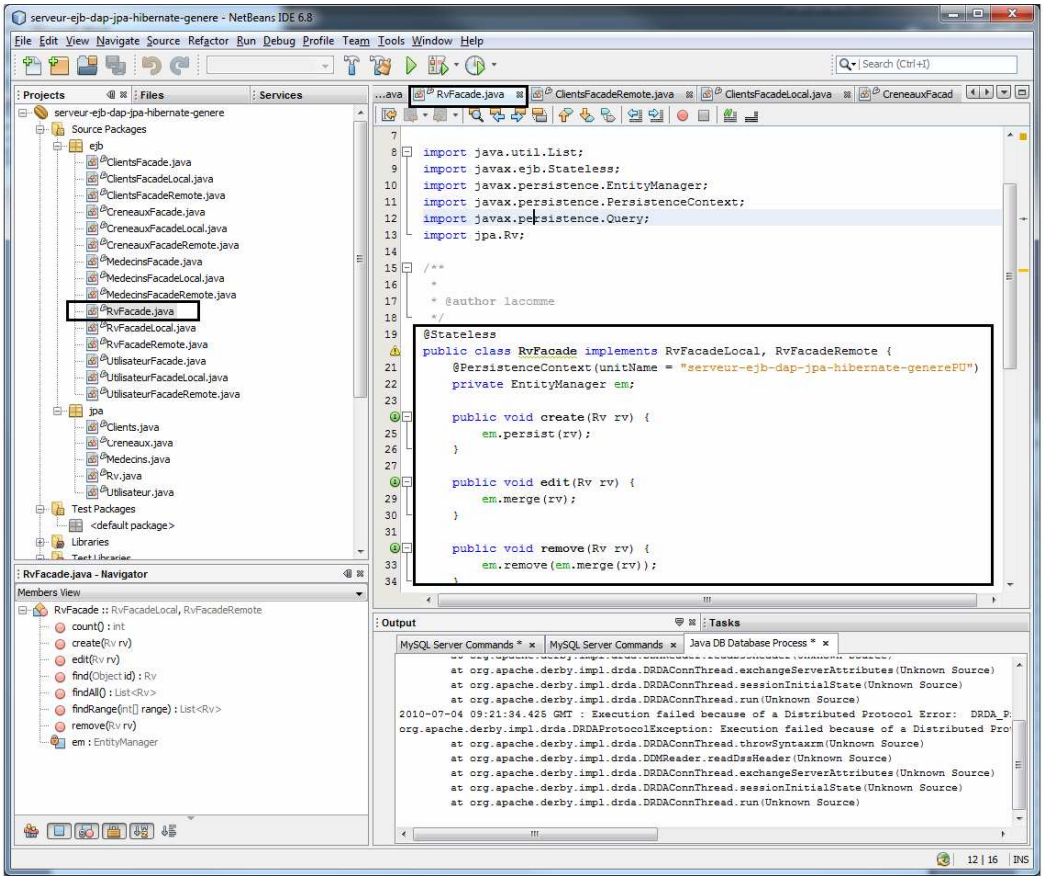

New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project:
EJBModule1
Categories:
Google Web Toolkit
Java Card 3 Platform
Java EE
Context and Dependency Inje
Java
JavaFX
JavaBeans Objects
JUnit
Persistence
Groovy
File Types:
Session Bean
Message-Driven Bean
Service Locator
Caching Service Locator
Session Beans For Entity Classes
Standard Deployment Descriptor
Description:
Creates Session EJB Facade based on Entity class. This template creates Session EJB for every Entity class with basic access methods.
< Back
Next >
Finish
Cancel
Help


New Session Beans For Entity Classes
Steps
1. Choose File Type
2. Entity Classes
3. Generated Session Beans
Entity Classes
Available Entity Classes:
jpa.Clients
jpa.Creneaux
jpa.Medecins
jpa.Rv
Add >
< Remove
Add All >>
<< RemoveAll
Selected Entity Classes:
Include Referenced Classes
Select entity classes.
< Back
Next >
Finish
Cancel
Help


New Session Beans For Entity Classes
Steps
1. Choose File Type
2. Entity Classes
3. Generated Session Beans
Entity Classes
Available Entity Classes:
Selected Entity Classes:
jpa.Clients
jpa.Creneaux
jpa.Medecins
jpa.Rv
Add >
< Remove
Add All >>
<< RemoveAll
Include Referenced Classes
< Back
Next >
Finish
Cancel
Help


New Session Beans For Entity Classes
Steps
1. Choose File Type
2. Entity Classes
3. Generated Session Beans
Generated Session Beans
Specify the location of new Session Bean classes
Project: EJBModule1
Location: Source Packages
Package: ejb
Created Files: <ClassName>Facade, <ClassName>FacadeLocal, <ClassName>FacadeRemote for each entity class.
Create Interfaces:
Remote
Local
< Back
Next >
Finish
Cancel
Help

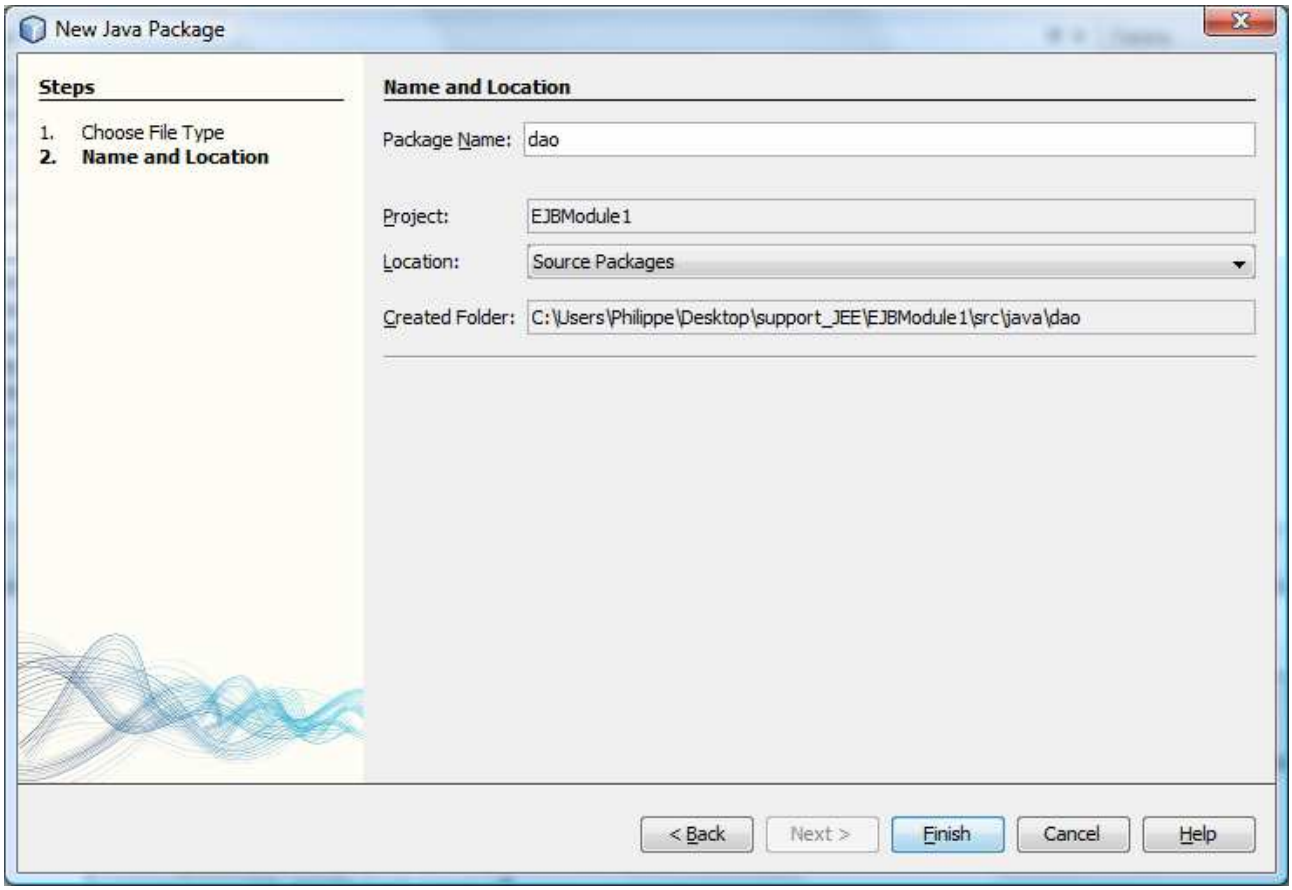




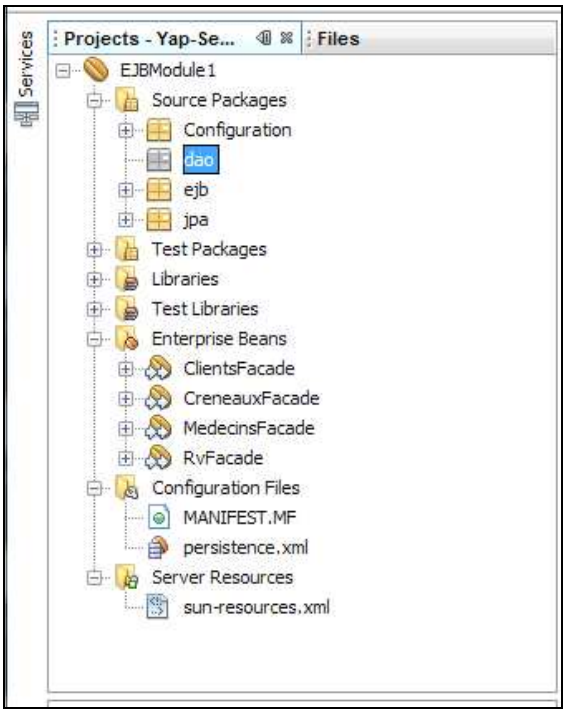
Cet ensemble de classes, contient des interfaces distantes et locales pour implémenter les méthodes du **CRUD**. Rappelons que **CRUD** désigne les quatre opérations de base pour la persistance des

données, en particulier le stockage d'informations en base de données. Soit : Create, Read (ou Retrieve), Update et Delete (ou Destroy).

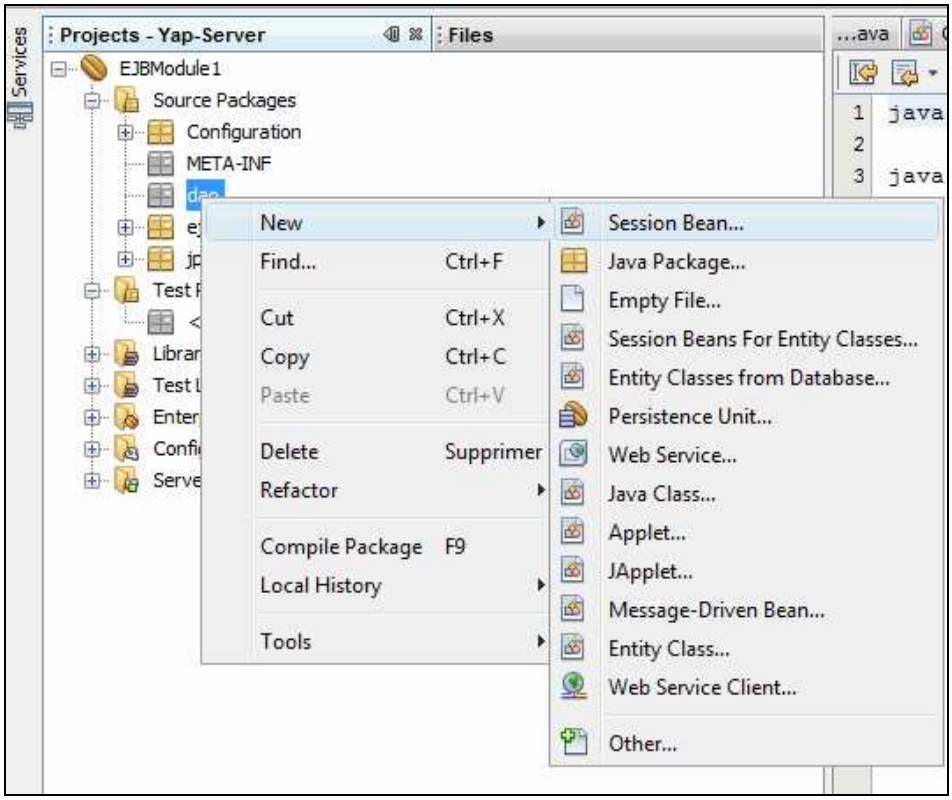
Enfin nous allons créer un package nommée **dao**, qui représente l'interface de l'EJB pour l'accès aux données.



Le projet se présente alors comme suit :



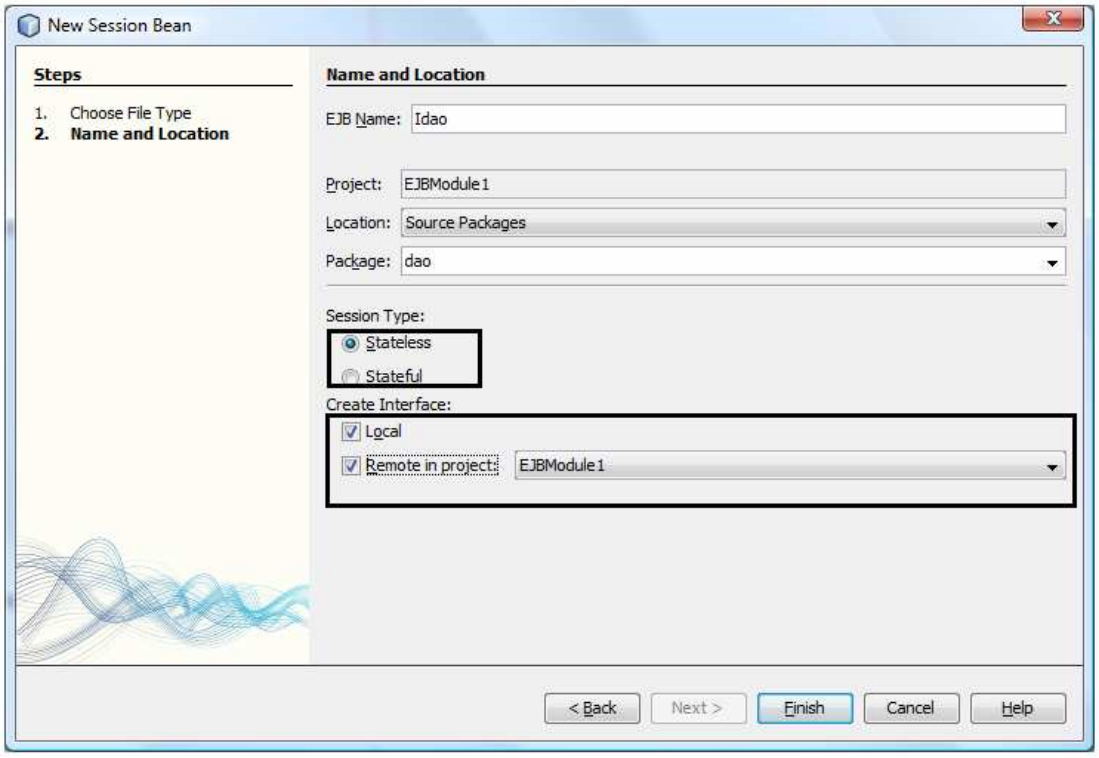
Ajoutons une **Session Bean** dans le package **dao**. Cette interface va permettre d'avoir plusieurs méthodes sur une seule interface (contrairement aux classes du package ejb qui sont en fait une interface par entité).



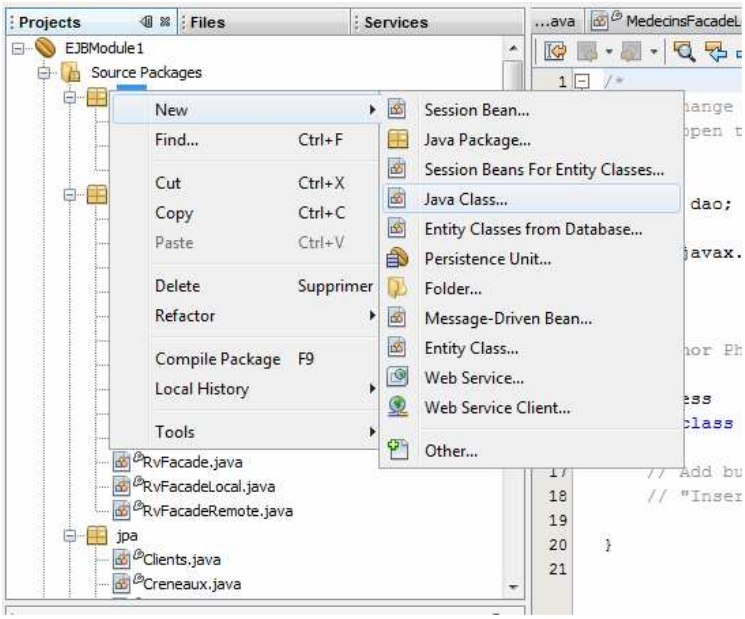
Demandons en plus une interface locale et distante de type **stateless**.

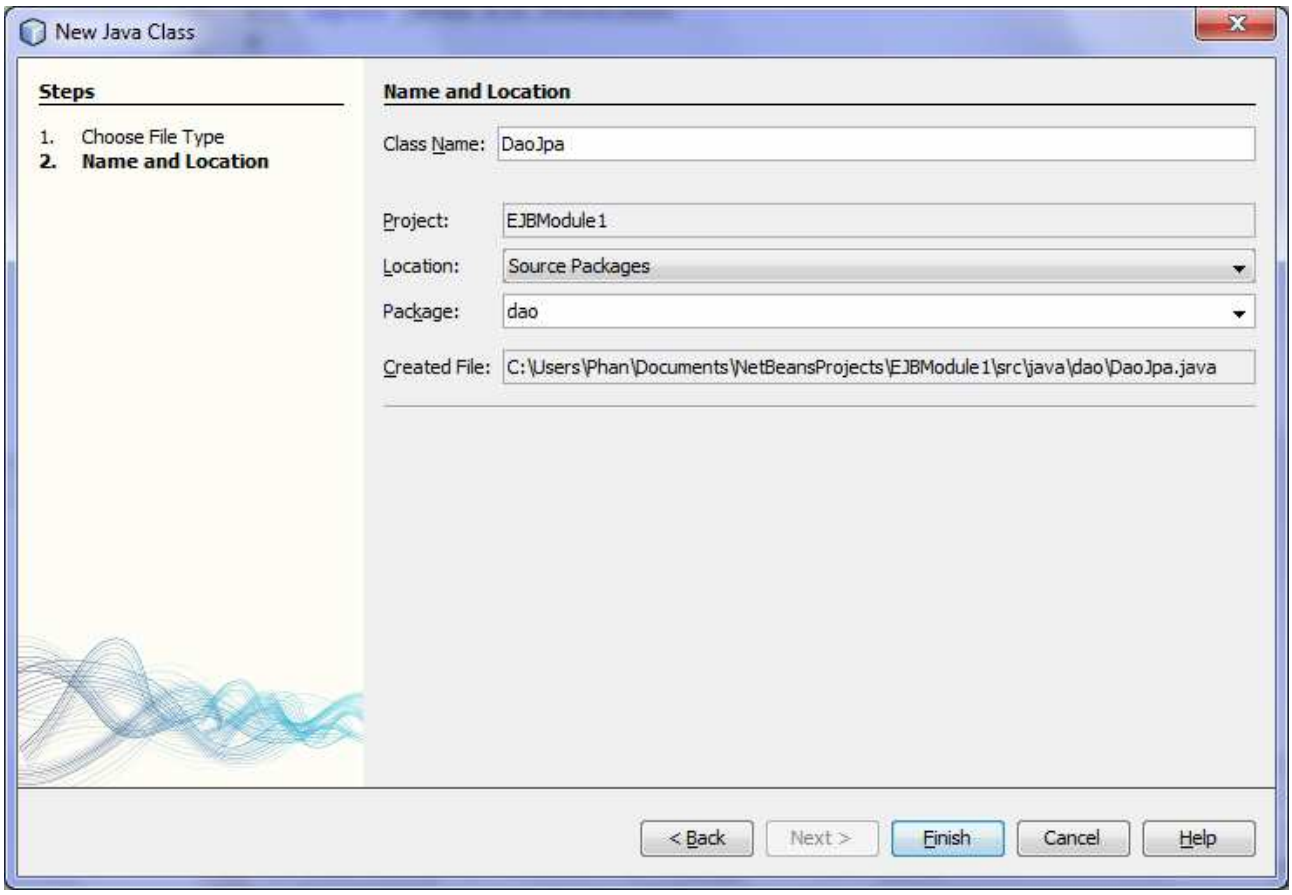
Rappels :

Stateless : L'état du Bean n'est pas conservé. Sa durée de vie correspond à la durée de vie d'une **requête** utilisateur (ex : Calculatrice).
Stateful : L'état du Bean est conservé. Sa durée de vie correspond à la durée de vie d'une **session** utilisateur (ex : Panier d'achat).

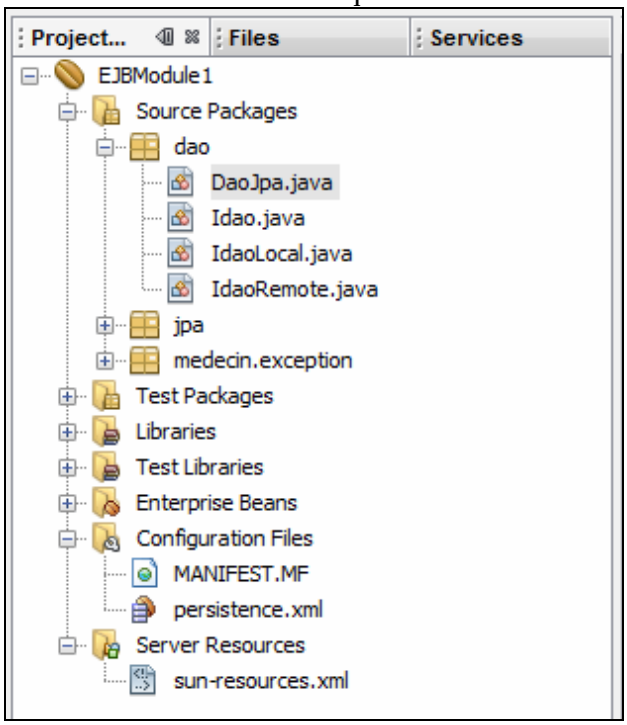


Nous allons complexifier un peu ce schéma :
- en ajoutant une nouvelle classe java **DaoJpa** qui contiendra une implémentation de **IdaoLocal** et **IdaoRemote**
- en considérant que le fichier **Idao** est juste une interface.





Ainsi le package Dao contient 4 fichiers comme indiqués ci-dessous :



Le fichier Idao.java

```
package dao;

import java.util.*;
import jpa.*;


public interface Idao {

 // liste des clients
 public List<Clients> getAllClients();
 // liste des Médecins
 public List<Medecins> getAllMedecins();
 // liste des créneaux horaires d'un médecin
 public List<Creneaux> getAllCreneaux(Medecins medecin);
 // liste des Rv d'un médecin, un jour donné
 public List<Rv> getRvMedecinJour(Medecins medecin, String jour);
 // trouver un client identifié par son id
 public Clients getClientById(Long id);
// trouver un client idenbtifié par son id
 public Medecins getMedecinById(Long id);
 // trouver un Rv identifié par son id
 public Rv getRvById(Long id);
 // trouver un créneau horaire identifié par son id
 public Creneaux getCreneauById(Long id);
 // ajouter un RV
 public Rv ajouterRv(String jour, Creneaux creneau, Clients client);
 // supprimer un RV
 public void supprimerRv(Rv rv);
}
```

IdaoLocal.java

```
package dao;

import javax.ejb.Local;

 @Local
 public interface IdaoLocal extends Idao{
}
```

IdaoRemote.java

```
package dao;

 import javax.ejb.Remote;

 @Remote
  public interface IdaoRemote extends Idao {
 }
```

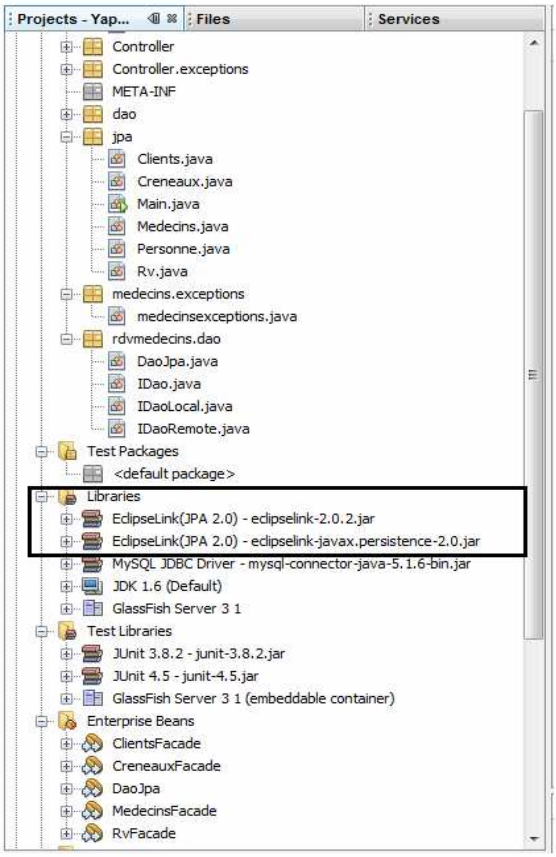
DaoJpa.java

```
package dao;

import javax.ejb.Stateless;
import javax.*;
import jpa.*;
import java.util.*;
import javax.ejb.*;
import javax.persistence.*;
import java.text.*;
//import javax.transaction.Transaction;
import medecin.exception.*;

@Stateless(mappedName = "Interface")
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class DaoJpa implements IdaoLocal, IdaoRemote {

    @PersistenceContext
    private EntityManager em;
```

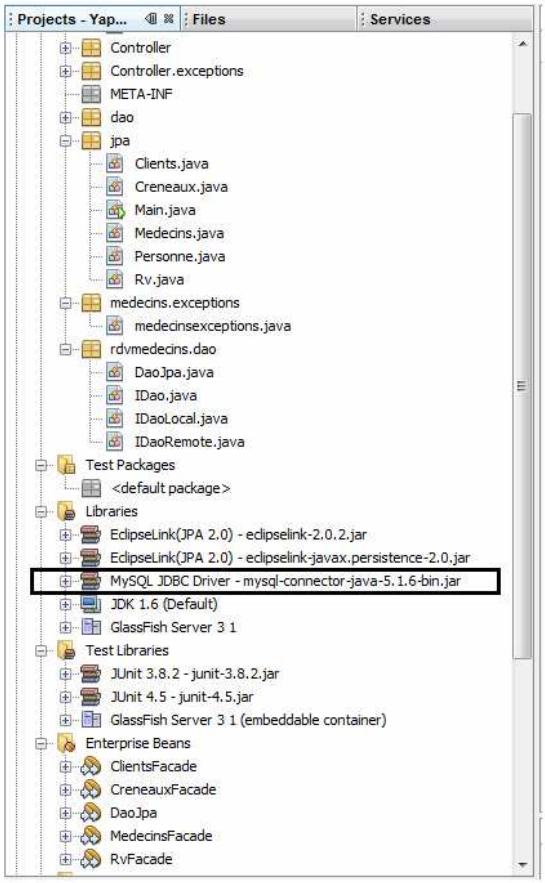
```
    // liste des clients
    public List<Clients> getAllClients() {
        try {
            return em.createQuery("select c from Clients c").getResultList();
        } catch (Throwable th) {
            throw new MedecinException (th, 1);
        }
    }

    // liste des médecins
    public List<Medecins> getAllMedecins() {
        try {
            return em.createQuery("select m from Medecins m").getResultList();
        } catch (Throwable th) {
            throw new MedecinException (th, 2);
        }



    }

    // liste des créneaux horaires d'un médecin donné
    // medecin : le médecin
    public List<Creneaux> getAllCreneaux(Medecins medecin) {
        try {
            return em.createQuery("select c from Creneaux c join c.medecin m where
m.id=:idMedecin").setParameter("idMedecin", medecin.getId()).getResultList();
        } catch (Throwable th) {
            throw new MedecinException (th, 3);
        }
    }

    // liste des Rv d'un médecin donné, un jour donné
    // medecin : le médecin
    // jour : le jour
    public List<Rv> getRvMedecinJour(Medecins medecin, String jour) {
        try {
            return em.createQuery("select rv from Rv rv join rv.creneau c join c.medecin m where
m.id=:idMedecin and rv.jour=:jour").setParameter("idMedecin", medecin.getId()).setParameter("jour",
new SimpleDateFormat("yyyy:MM:dd").parse(jour)).getResultList();
        } catch (Throwable th) {
            throw new MedecinException (th, 4);
        }
    }

    // ajout d'un Rv
    // jour : jour du Rv
    // creneau : créneau horaire du Rv
    // client : client pour lequel est pris le Rv
    public Rv ajouterRv(String jour, Creneaux creneau, Clients client) {
        try {
            Rv rv = new Rv(new SimpleDateFormat("yyyy:MM:dd").parse(jour), client, creneau);
            em.persist(rv);
            return rv;
        } catch (Throwable th) {
            throw new MedecinException (th, 5);
        }
    }

    // suppression d'un Rv
    // rv : le Rv supprimé
    public void supprimerRv(Rv rv) {
        try {
            em.remove(em.merge(rv));
        } catch (Throwable th) {
            throw new MedecinException (th, 6);
        }
    }

    // récupérer un client donné
    public Clients getClientById(Long id) {
        try {
            return (Clients) em.find(Clients.class, id);
        } catch (Throwable th) {
            throw new MedecinException (th, 7);
        }
    }

    // récupérer un médecin donné
    public Medecins getMedecinById(Long id) {
```

```
        try {
            return (Medecins) em.find(Medecins.class, id);
        } catch (Throwable th) {
            throw new MedecinException (th, 8);
        }
    }

    // récupérer un Rv donné
    public Rv getRvById(Long id) {
        try {
            return (Rv) em.find(Rv.class, id);
        } catch (Throwable th) {
            throw new MedecinException (th, 9);
        }
    }

    // récupérer un créneau donné
    public Creneaux getCreneauById(Long id) {
        try {
            return (Creneaux) em.find(Creneaux.class, id);
        } catch (Throwable th) {
            throw new MedecinException (th, 10);
        }
    }
}
```

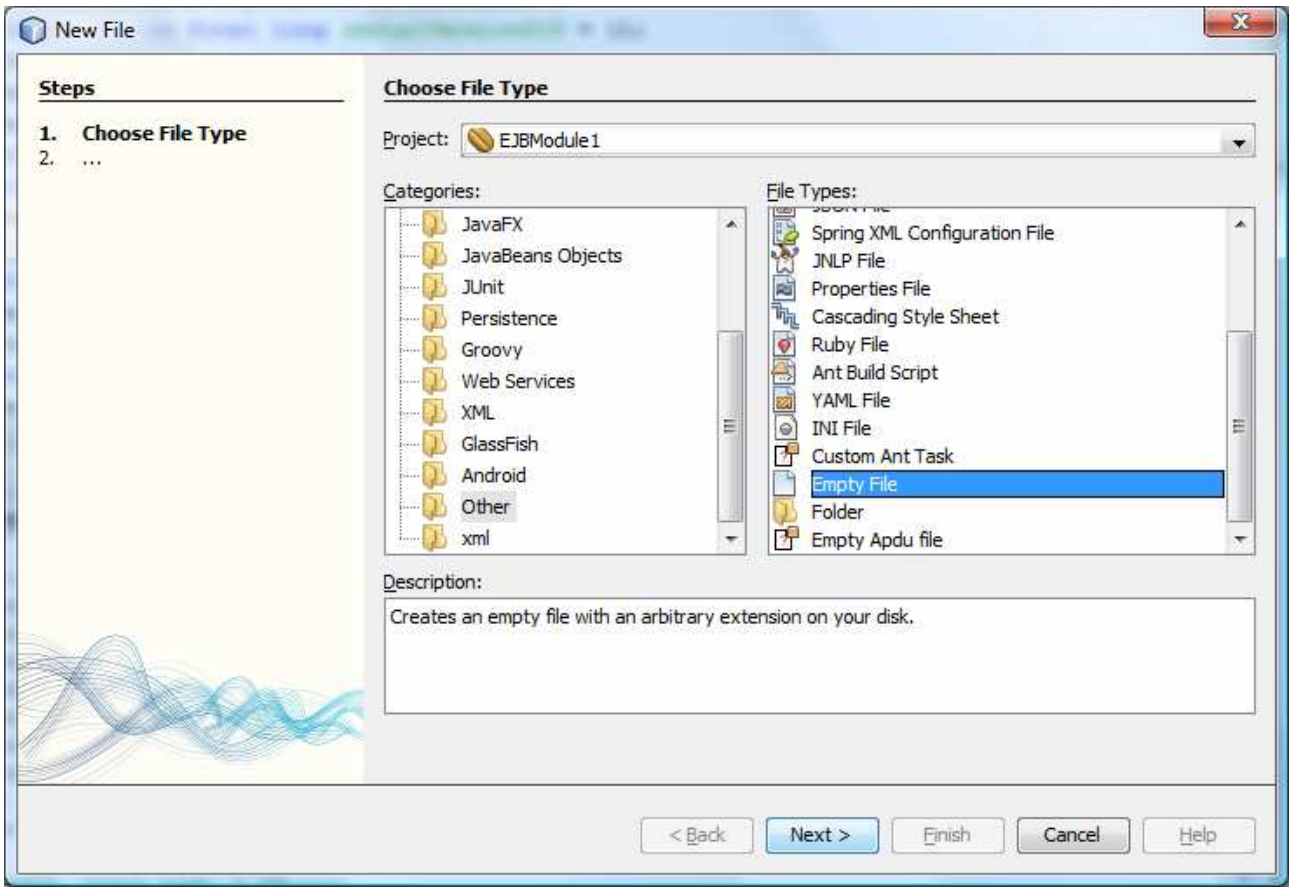
Avant de compiler assurez-vous que les librairies EclipseLink sont incluses dans le projet :



D'autre part, il faut inclure la librairie MySQL :



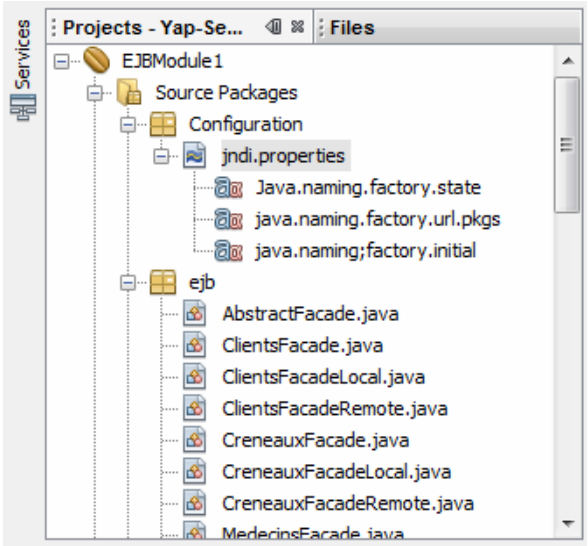
Enfin il faut créer un fichier de configuration de JNDI nommé : jndi.properties.
Le plus simple est de créer un package nommé **Configuration**.
Ensuite il suffit de faire **Clic Droit / New / Others / empty File**



Le fichier créé doit contenir le texte suivant :

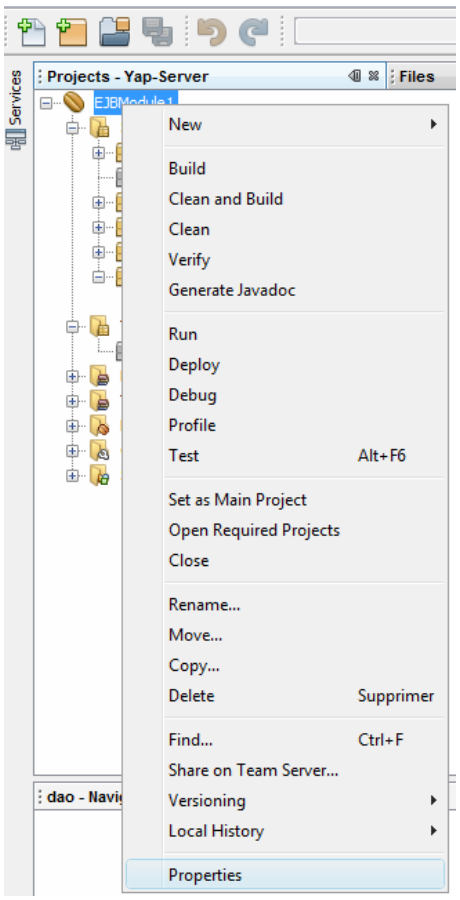
```
java.naming;factory.initial = com.sun.enterprise.naming.SerialInitContextFactory
java.naming.factory.url.pkgs = com.sun.enterprise.naming
java.naming.factory.state=com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl
```

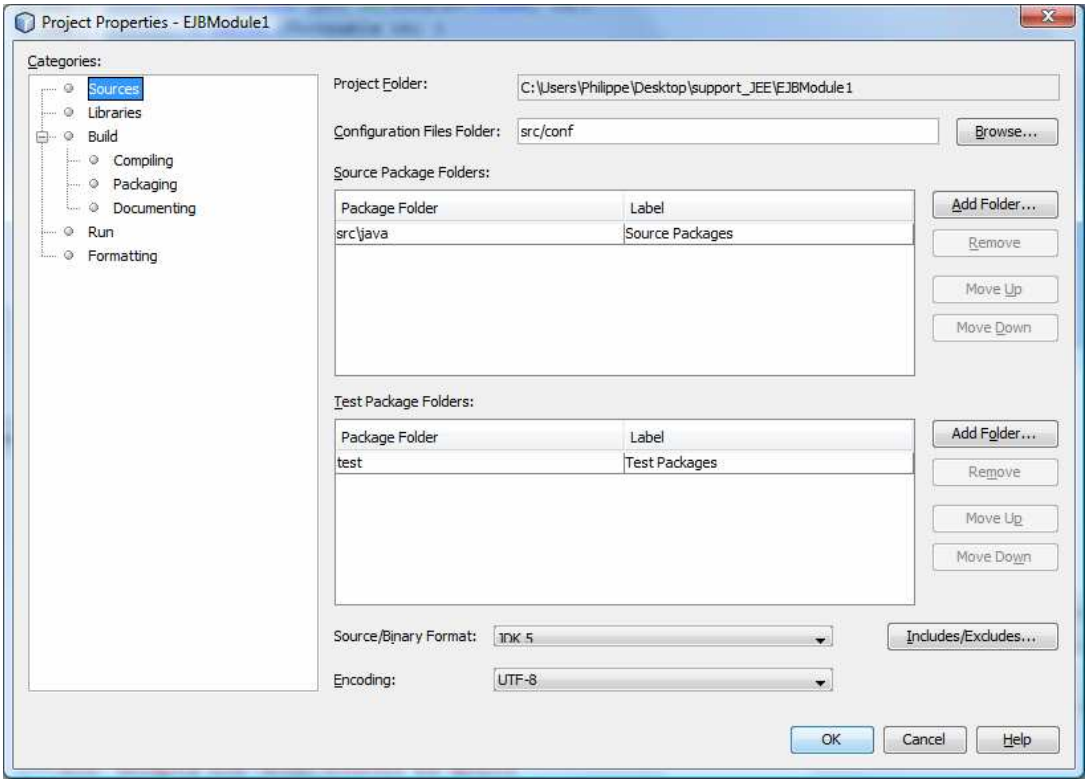
Finalement, dans le projet, on obtient :



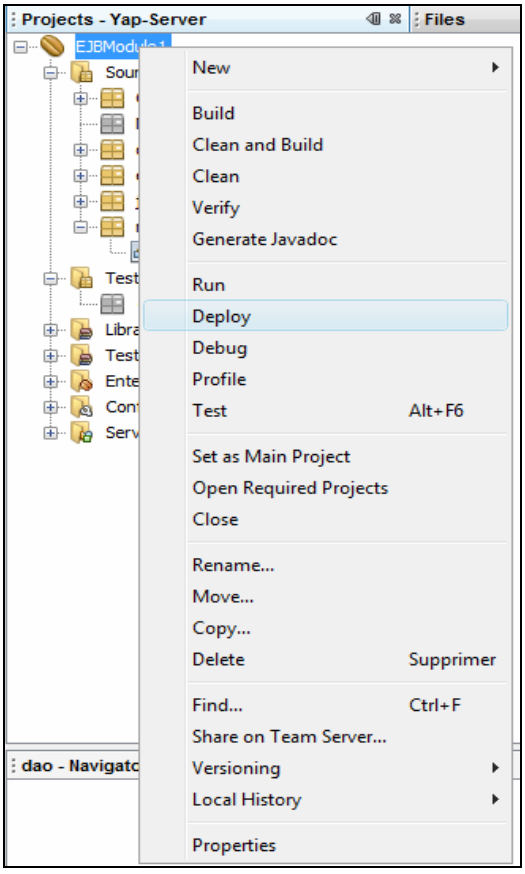
4) Déploiement de l'EJB

4.1. Vérifier les propriétés : Clic Droit, Properties

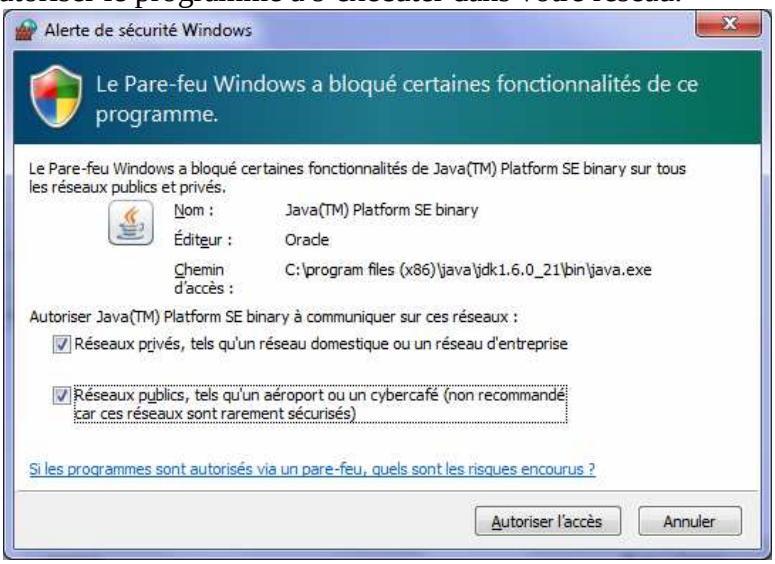




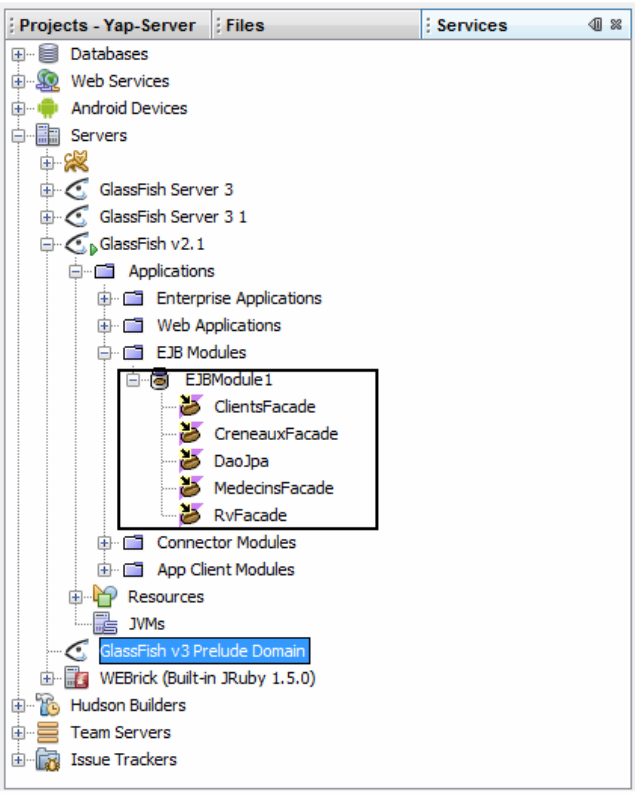
4.3. Déployer



Il faudra peut-être autoriser le programme à s'exécuter dans votre réseau.



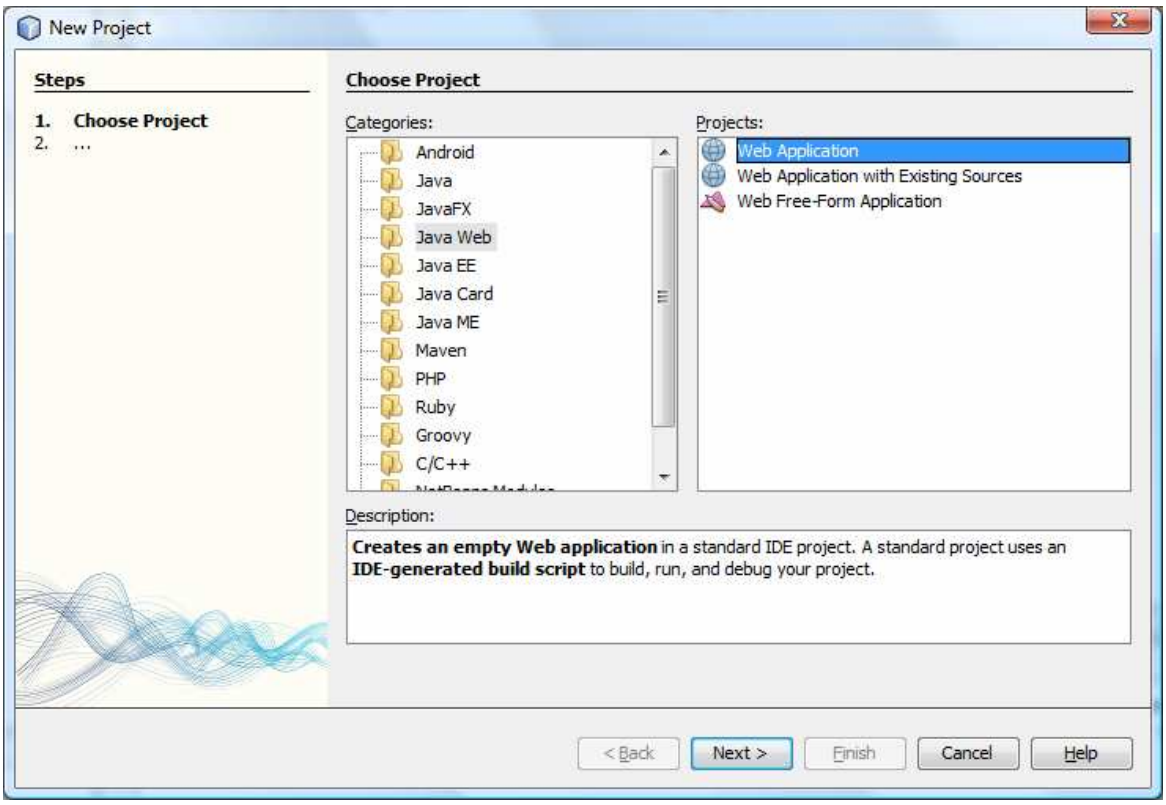
Ensuite, le déploiement a réussi et on peut constater que l'EJB a été ajouté à la partie **Applications** du serveur **Glassfish**.


Projects - Yap-Server
Files
Services
Databases
Web Services
Android Devices
Servers
GlassFish Server 3
GlassFish Server 3 1
GlassFish v2.1
Applications
Enterprise Applications
Web Applications
EJB Modules
EJBModule1
ClientsFacade
CreneauxFacade
DaoJpa
MedecinsFacade
RvFacade
Connector Modules
App Client Modules
Resources
JVMs
GlassFish v3 Prelude Domain
WEBrick (Built-in JRuby 1.5.0)
Hudson Builders
Team Servers
Issue Trackers

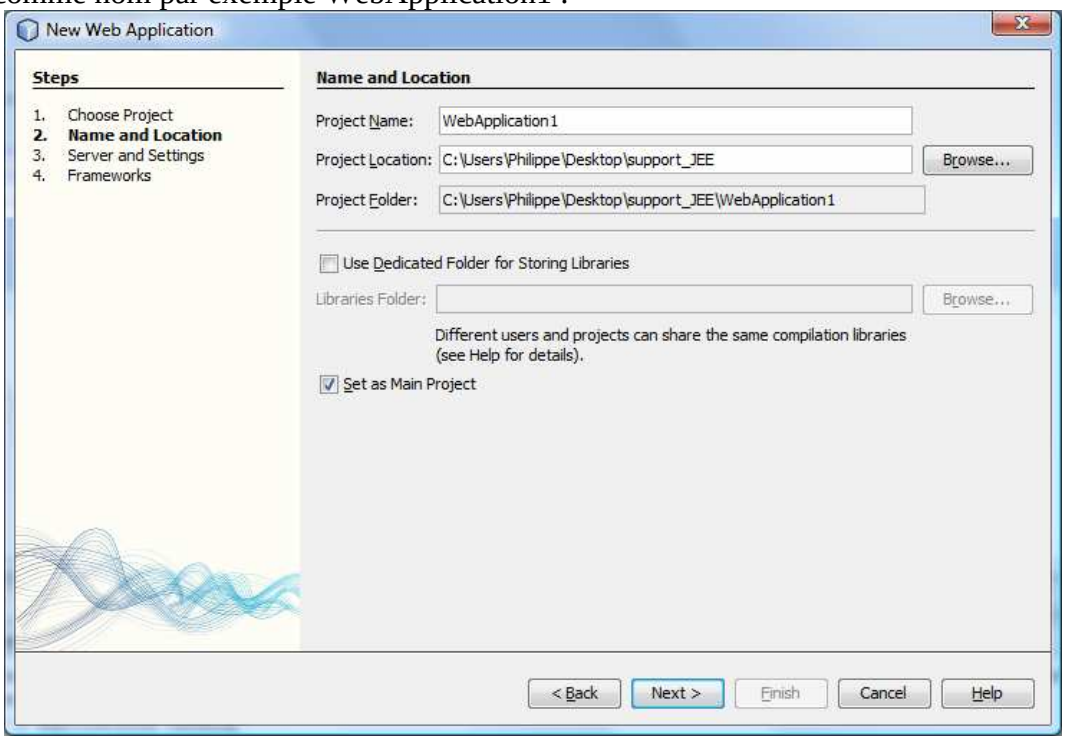
Partie 3. Création d'un Web Service

3.1. Création du projet

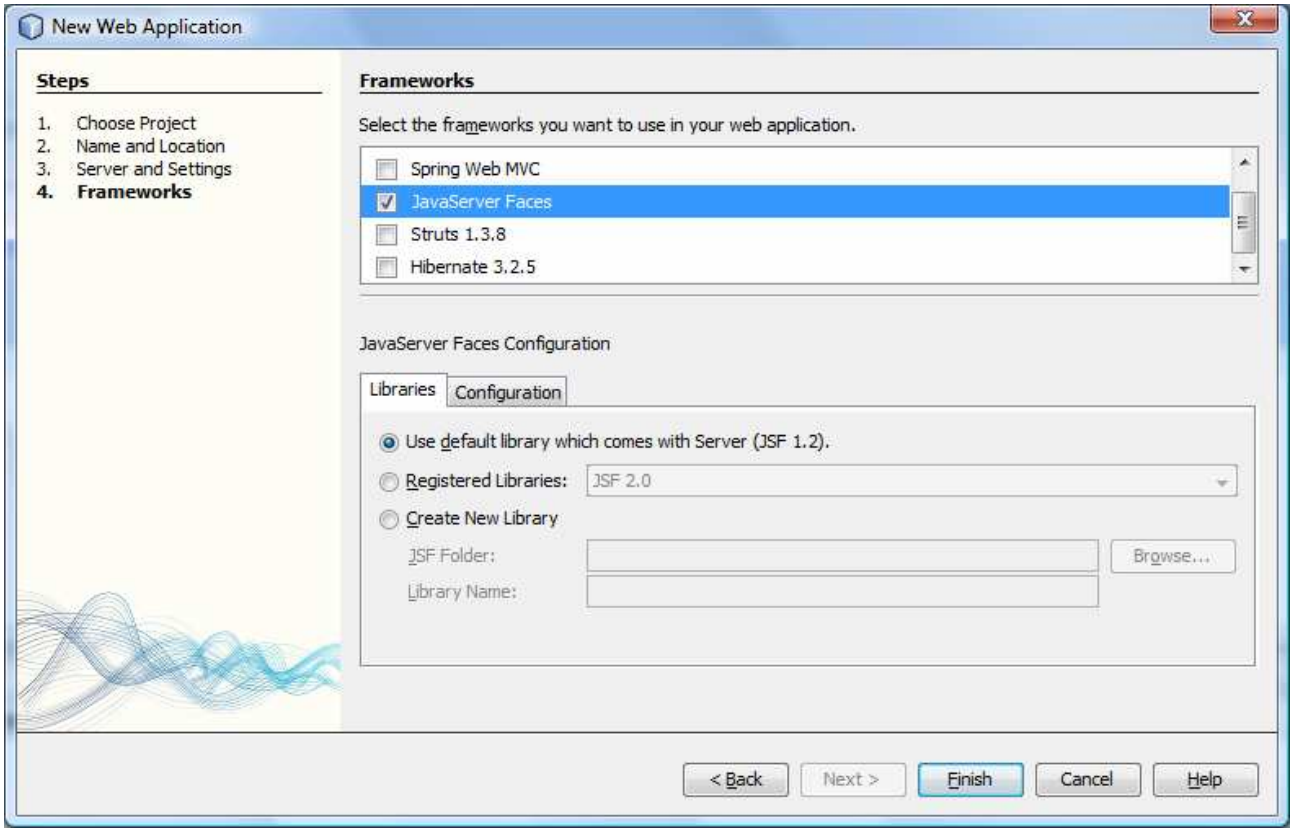
Créer une application web. Menu **File, New Project**.



Choisir comme nom par exemple WebApplication1 :

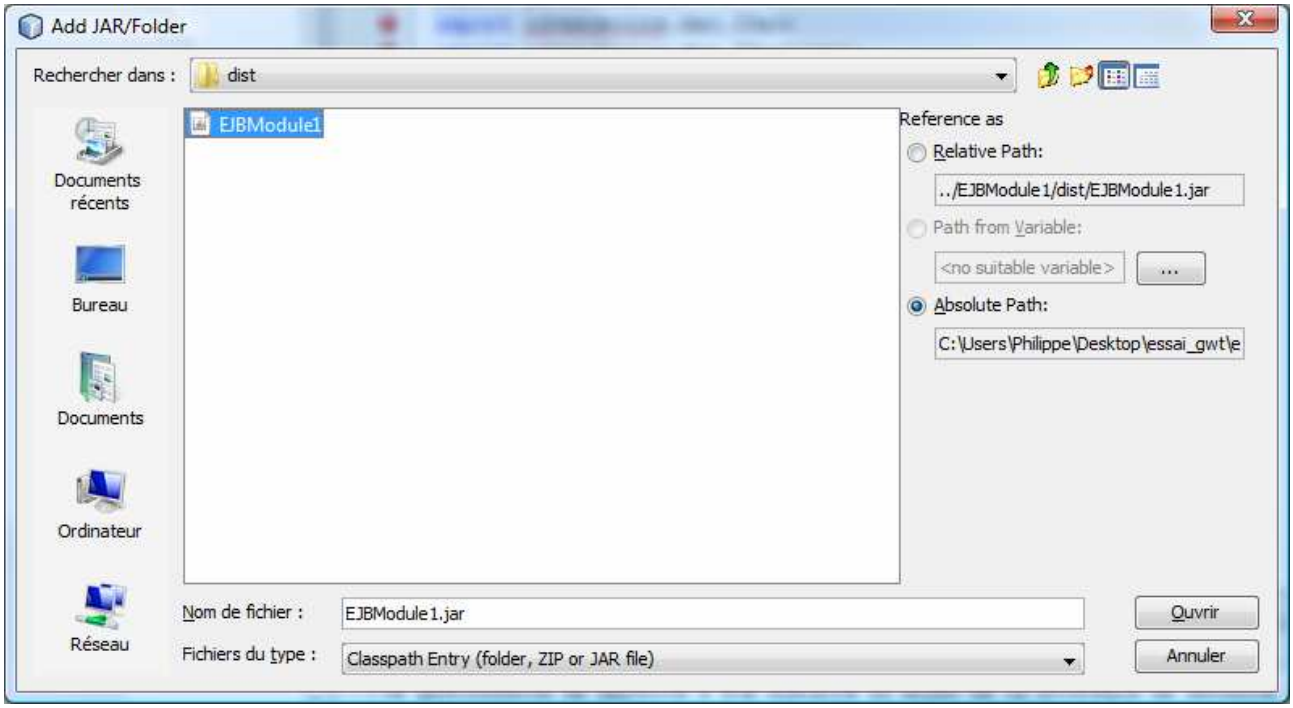


Parmi les frameworks installés vous pouvez par exemple choisir Java Server Faces ou ne rien choisir (il n'y aura pas d'exemple d'utilisation du framework dans ce tutorial).

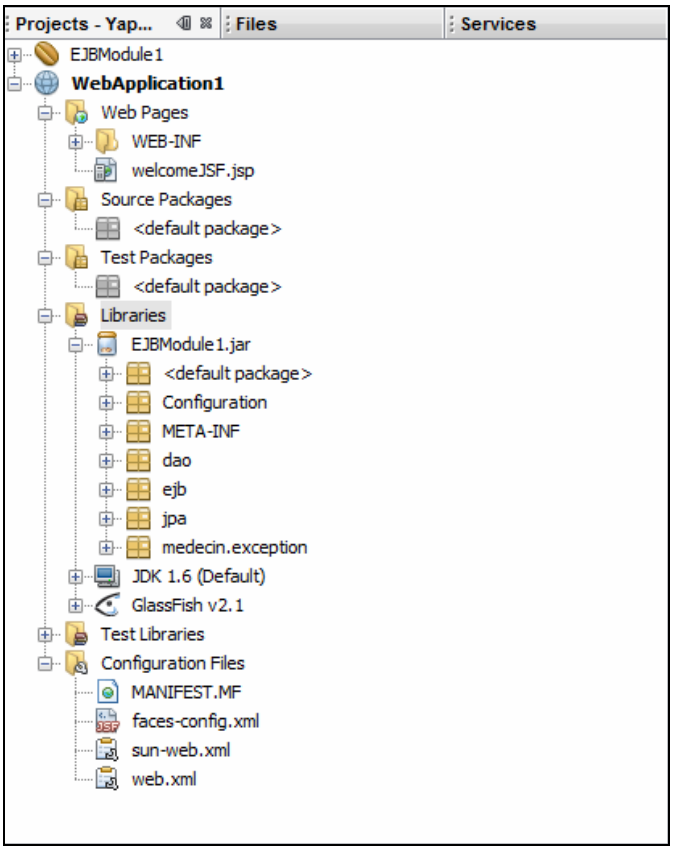


3.2. Utilisation de l'EJB

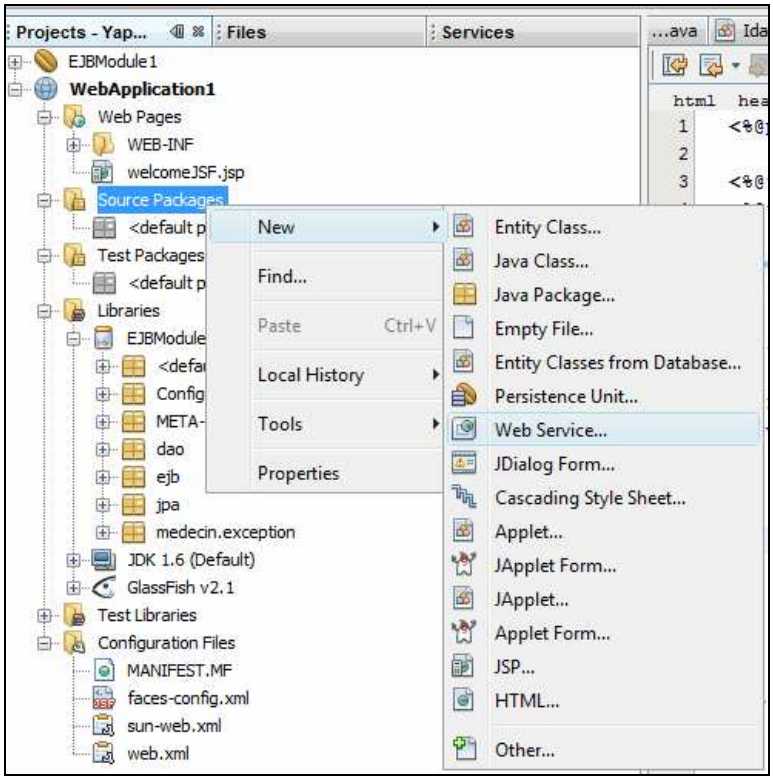
Faire un clic droit sur la partie « librairies » de l'application Web et choisir « Add Jar »
Choisir ensuite EJBModule1.jar.



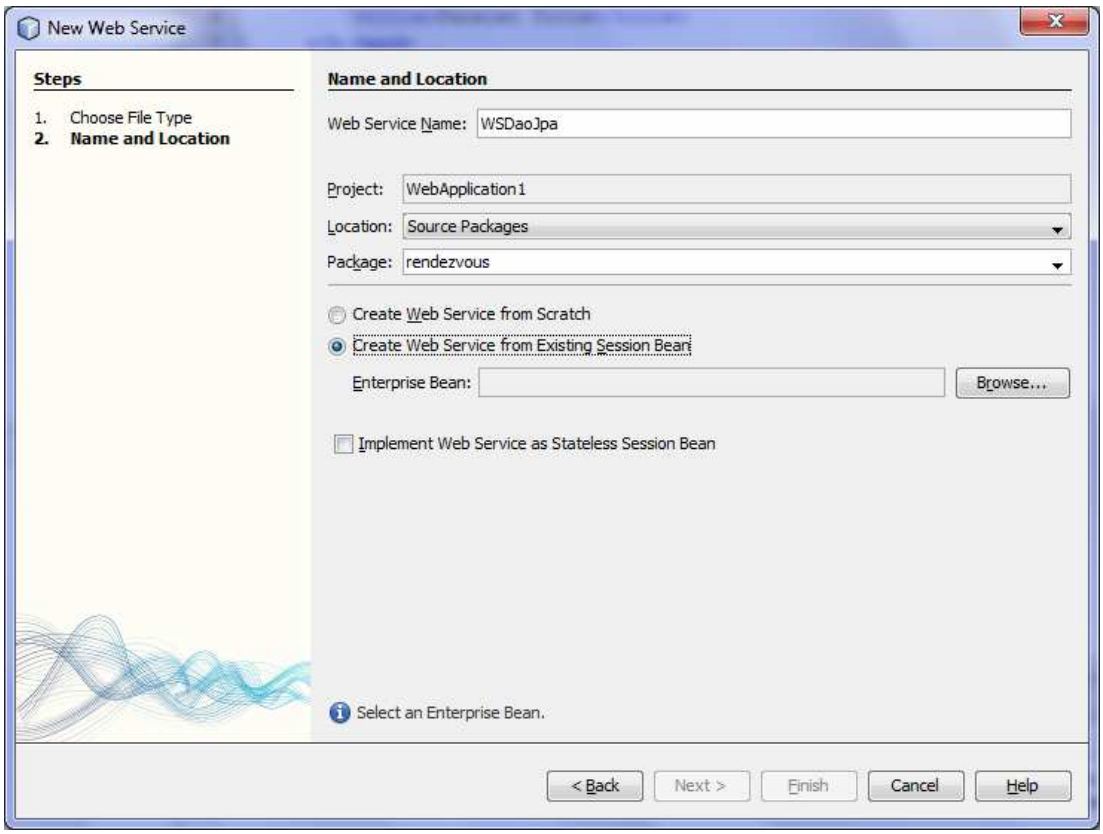
Le projet WebApplication1 devrait se présenter comme suit :



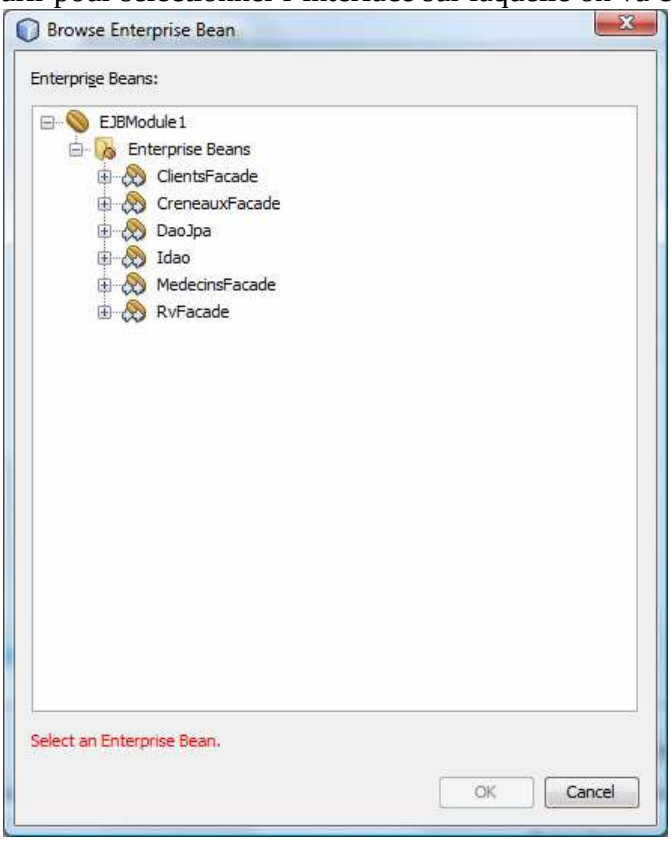
Nous allons créer le web service en faisant **new / Web Service.**



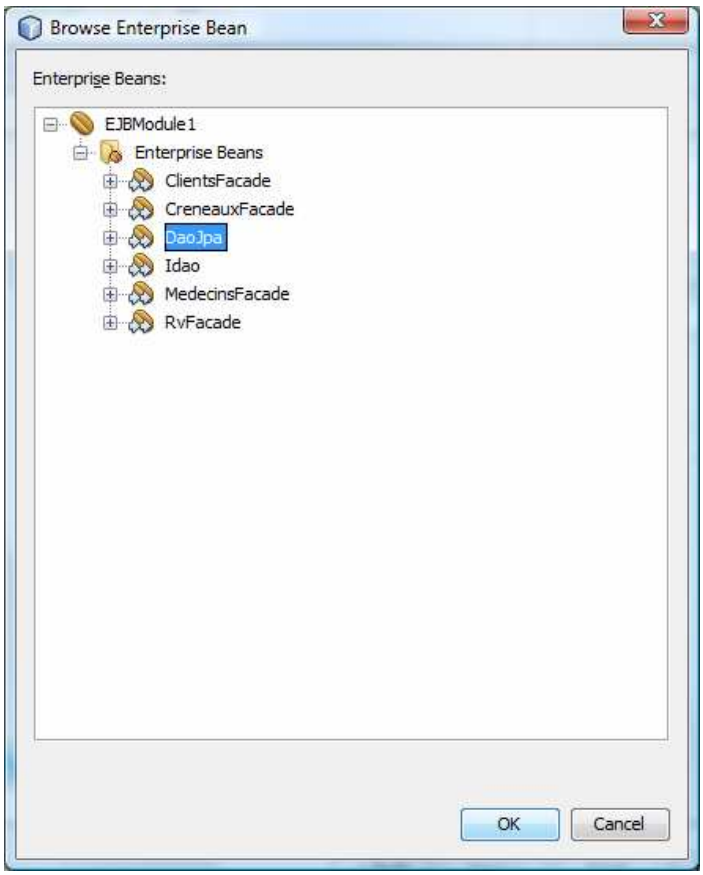
Il est possible de stocker cette classe dans un package r**endezvous** par exemple.



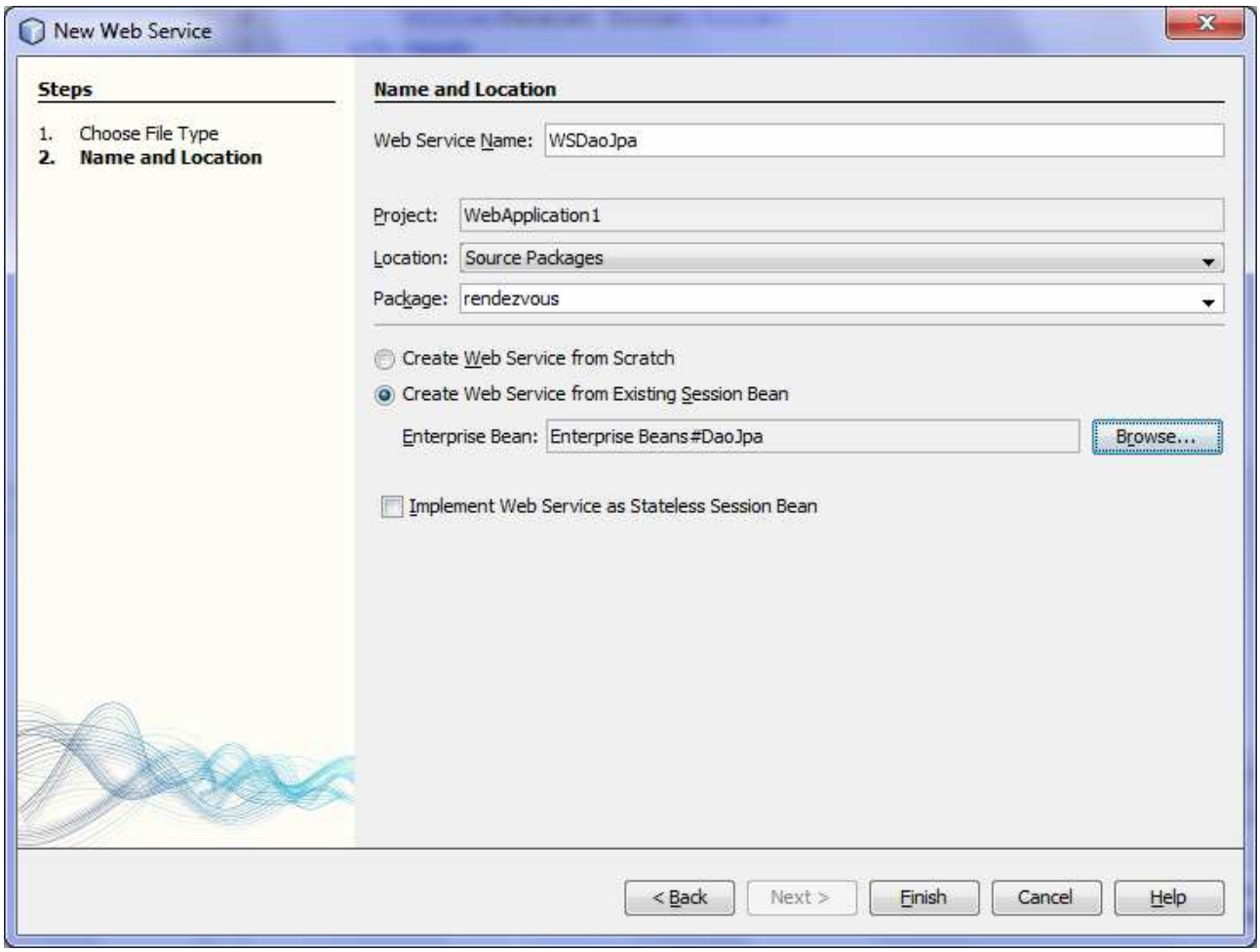
Utilisez le bouton parcourir pour sélectionner l'interface sur laquelle on va créer le service web.



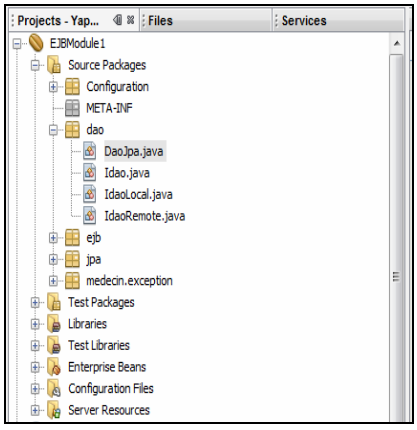
Sélectionner DaoJpa.



Utilisons comme nom **WSDaoJpa**.
Valider le choix pour revenir à la définition du web service.



Pour créer le web service, nous allons faire une couche devant la partie DaoJpa de l'EJB. Rappelons que la partie **DaoJpa** contient un ensemble de méthodes contrôlant l'accès aux données.



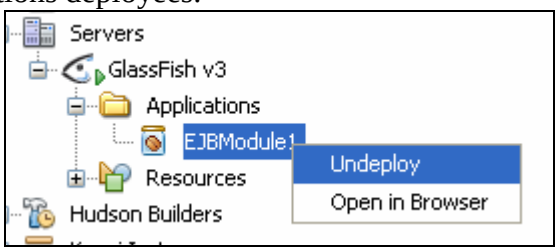

```
@Stateless(mappedName = "Interface")
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class DaoJpa implements IdaoLocal, IdaoRemote {

    @PersistenceContext
    private EntityManager em;

    // liste des clients
    public List<Clients> getAllClients() {
        try {
            return em.createQuery("select c from Clients c").getResultList();
        } catch (Throwable th) {
            throw new medecinsexceptions(th, 1);
        }
    }

    // liste des médecins
    public List<Medecins> getAllMedecins() {
        try {
            return em.createQuery("select m from Medecins m").getResultList();
        } catch (Throwable th) {
            throw new medecinsexceptions(th, 2);
```

A titre d'exemple, le web service peut se présenter comme suit :

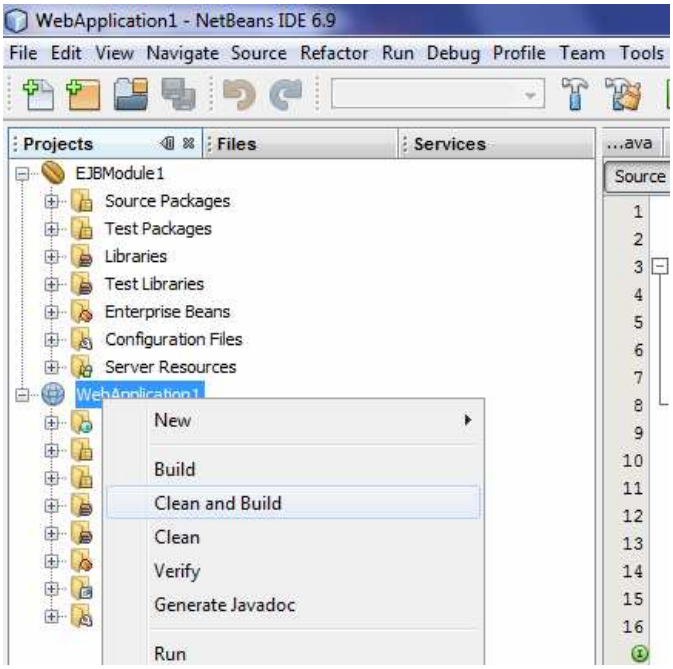
```
package rendezvous;

import javax.ejb.EJB;
import javax.jws.*;
import jpa.*;
import dao.*;
import dao.IdaoLocal;
import java.util.*;

@WebService()
public class WSDaoJpa implements Idao {
    @EJB
    private IdaoLocal dao;

   // web service numero 1
  // liste des clients
  @WebMethod
  public List<Clients> getAllClients() {
     return dao.getAllClients();
  }

  // web service numero 2
  // liste des clients
  @WebMethod
  public List<Medecins> getAllMedecins() {
    return dao.getAllMedecins();
 }

 // liste des créneaux horaires d'un médecin donné
 // medecin : le médecin
 @WebMethod
 public List<Creneaux> getAllCreneaux(Medecins medecin) {
   return dao.getAllCreneaux(medecin);
 }

 // liste des Rv d'un médecin donné, un jour donné
 // medecin : le médecin,  jour : le jour
 @WebMethod
 public List<Rv> getRvMedecinJour(Medecins medecin, String jour) {
    return dao.getRvMedecinJour(medecin, jour);
 }

 // ajout d'un Rv,  jour : jour du Rv
 // creneau : créneau horaire du Rv, client : client pour lequel est pris le Rv
 @WebMethod
 public Rv ajouterRv(String jour, Creneaux creneau, Clients client) {
    return dao.ajouterRv(jour, creneau, client);
}

// suppression d'un Rv, rv : le Rv supprimé
@WebMethod
public void supprimerRv(Rv rv) {
 dao.supprimerRv(rv);
}

// récupérer un client donné
@WebMethod
public Clients getClientById(Long id) {
 return dao.getClientById(id);
}

// récupérer un médecin donné
@WebMethod
 public Medecins getMedecinById(Long id) {
   return dao.getMedecinById(id);
}

// récupérer un Rv donné
@WebMethod
public Rv getRvById(Long id) {
 return dao.getRvById(id);
}

// récupérer un créneau donné
@WebMethod
public Creneaux getCreneauById(Long id) {
 return dao.getCreneauById(id);
}
    }
```

3.3. Test du web service

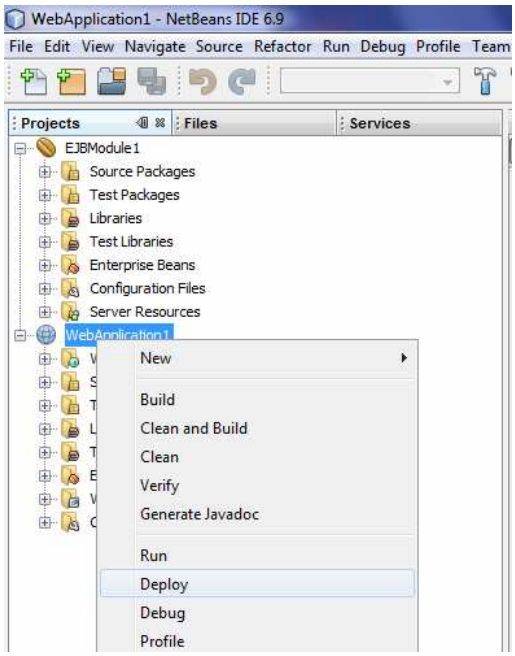
Vérifier que ni l'EJB ni la web application n'est déployé dans les services. Si ce n'est pas le cas, supprimer toutes les applications déployées.



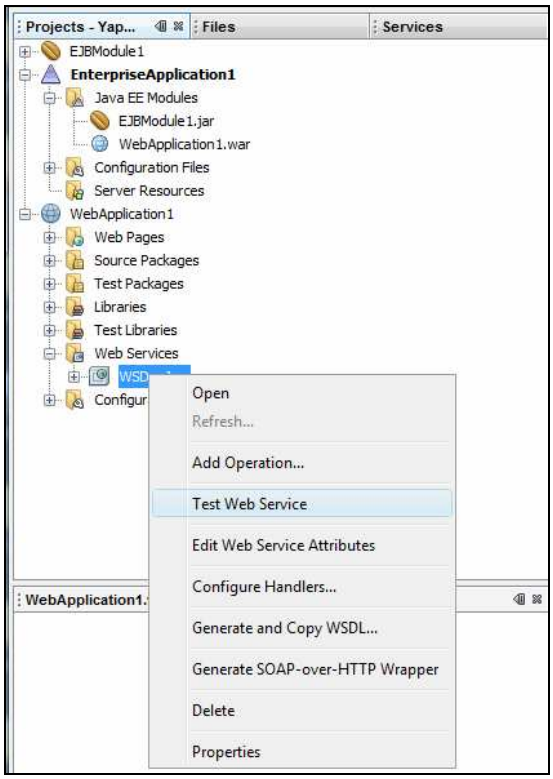
Compiler la web application.



Puis déployer la web application.

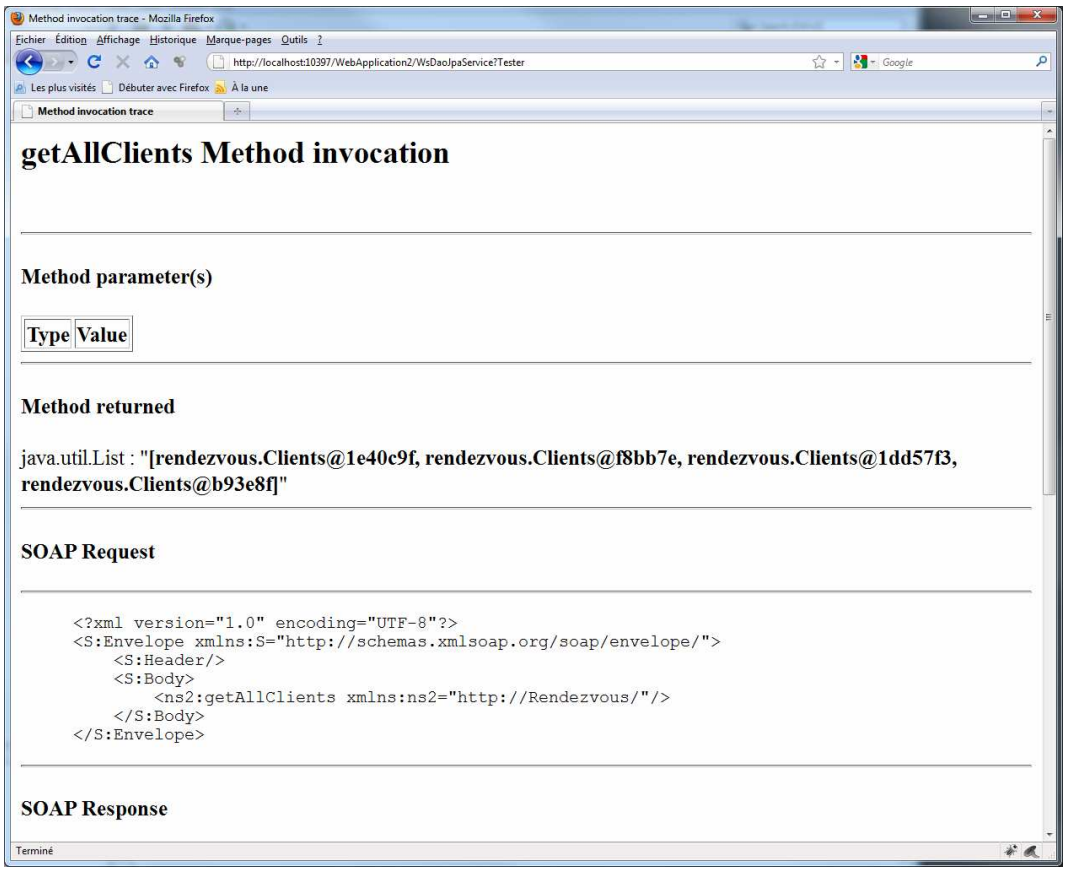


Puis, faire un clic droit sur **WsDaoJpa** dans le projet WebApplication1. Choisir **Test WebService.**

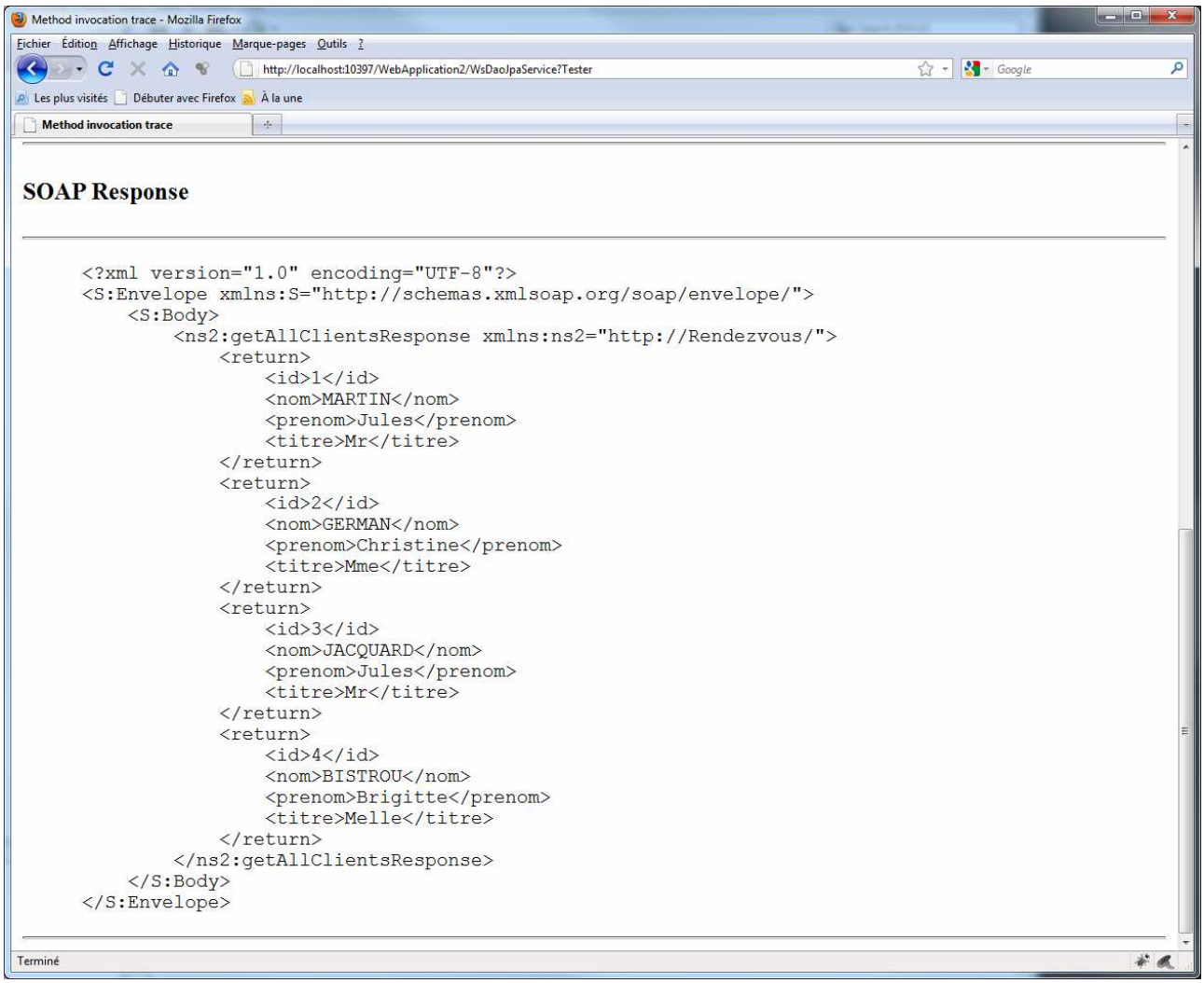


Ceci permet de tester les web services directement à partir d'un navigateur !

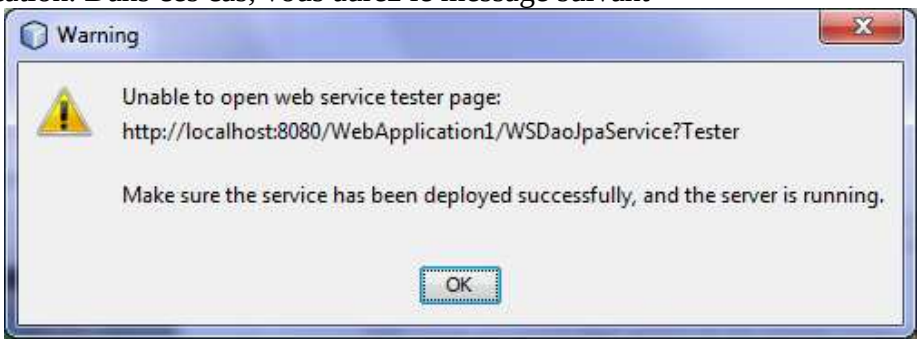

WSDaoJpaService Testeur de service Web - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://localhost:8080/WebApplication1/WSDaoJpaService?Tester
Search the web (Babylon)
Les plus visités Débuter avec Firefox À la une essai.html essai.html
WSDaoJpaService Testeur de service...
WSDaoJpaService Testeur de service Web
Ce formulaire vous permet de tester l'implémentation du service Web (Fichier WSDL)
Pour appeler une opération, renseignez les zones d'entrée des paramètres de la méthode, puis cliquez sur le bouton portant le nom de cette méthode.
Méthodes :
public abstract java.util.List rendezvous.WSDaoJpa.getAllClients()
getAllClients ()
public abstract java.util.List rendezvous.WSDaoJpa.getAllMedecins()
getAllMedecins ()
public abstract java.util.List rendezvous.WSDaoJpa.getAllCreneaux(rendezvous.Medecins)
getAllCreneaux ( )
public abstract java.util.List rendezvous.WSDaoJpa.getRvMedecinJour(rendezvous.Medecins,java.lang.String)
getRvMedecinJour (,)
Terminé



Avec la réponse SOAP :

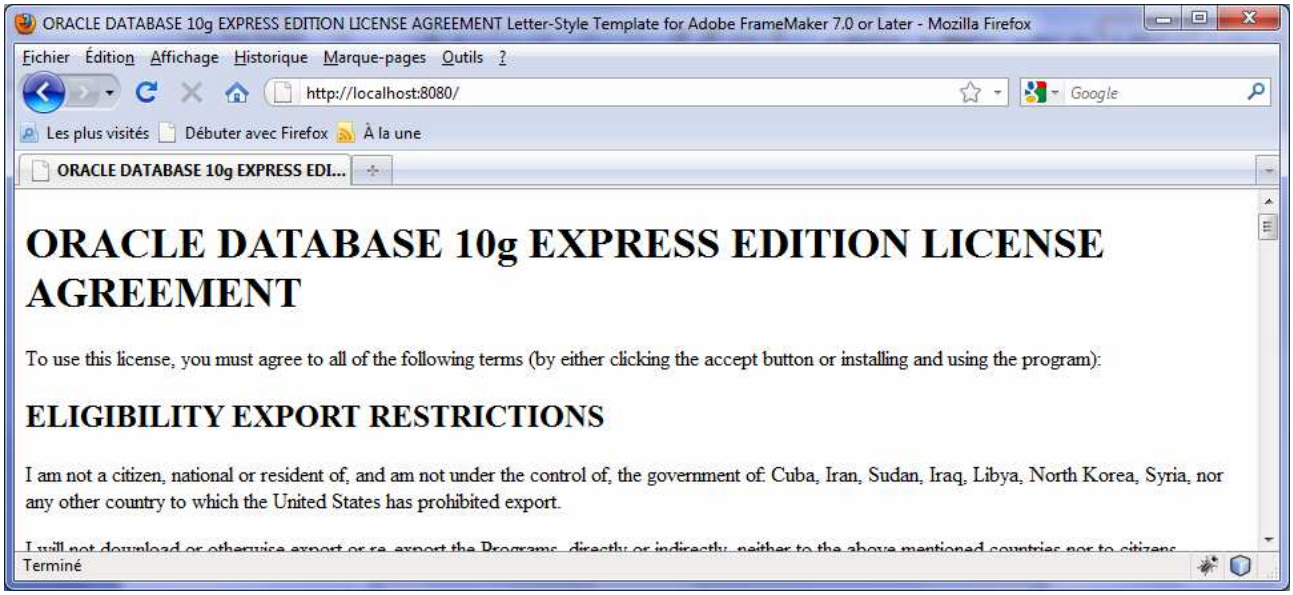


Il peut arriver que cette partie ne fonctionne pas soit parce que vous n'avez pas déployé correctement, soit parce que le port est déjà utilisé par une autre application. Dans ces cas, vous aurez le message suivant



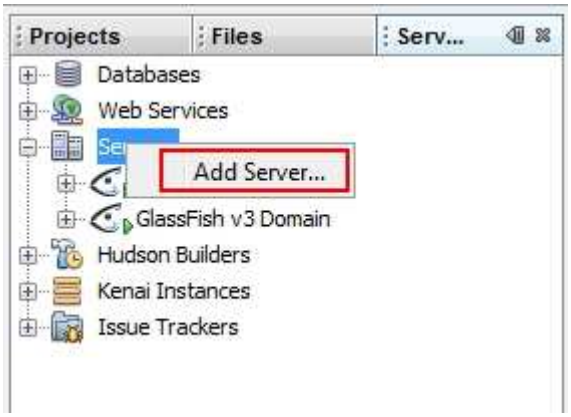
Si c'est un problème de déploiement, il faut vérifier tous les paramètres précédemment entrés. Si c'est un problème de port, il suffit de changer le port de glassfish.
Pour s'assurer que c'est bien un problème de port, on va entrer « localhost:8080 » (cf. message d'erreur) dans un navigateur.

Par exemple, nous avons obtenu ceci sur une machine de test :

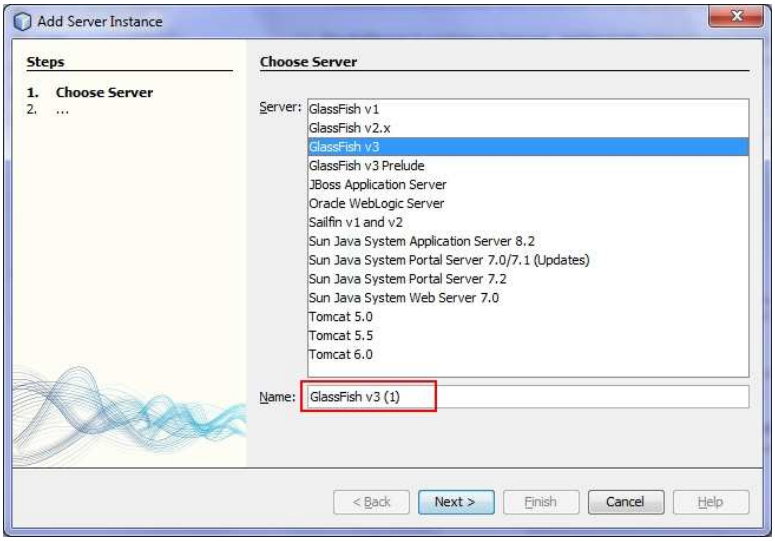


Cela indique que le port est occupé par oracle. Il faut donc dire à Glassfish d'utiliser un autre port.

Ajouter un nouveau serveur :

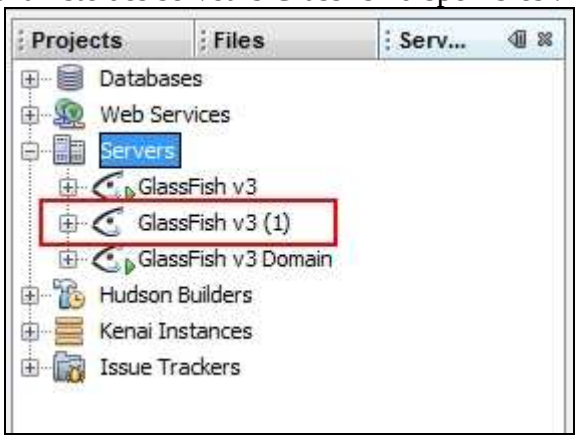


Et entrer un nom du nouveau domaine :

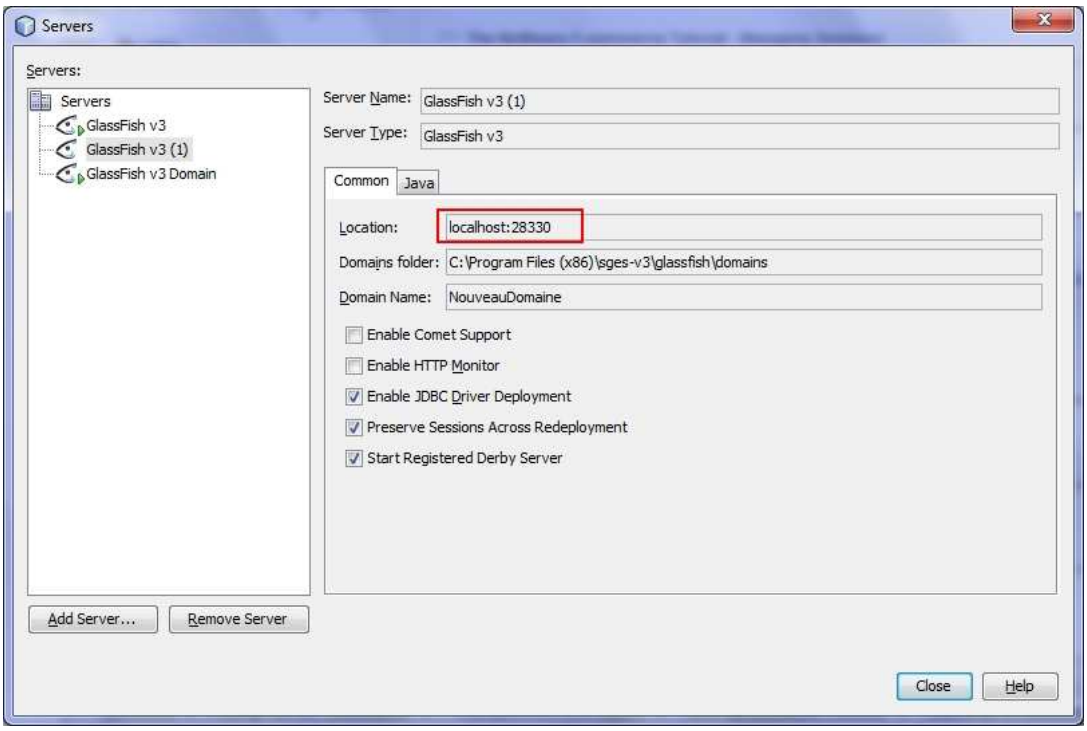


Puis valider les options par défaut.

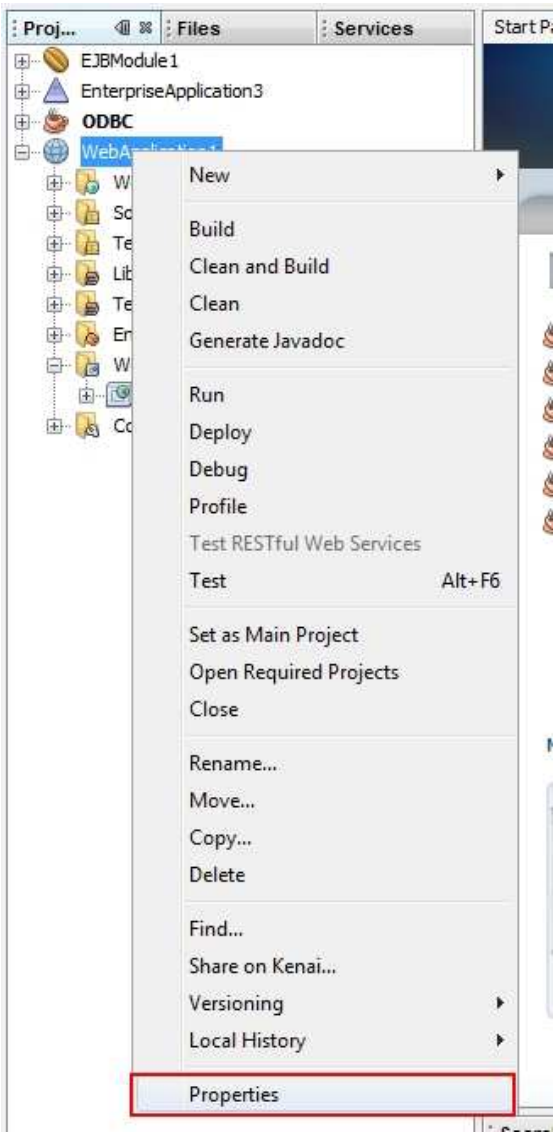
On retrouve ce serveur dans la liste des serveurs Glassfish disponibles :

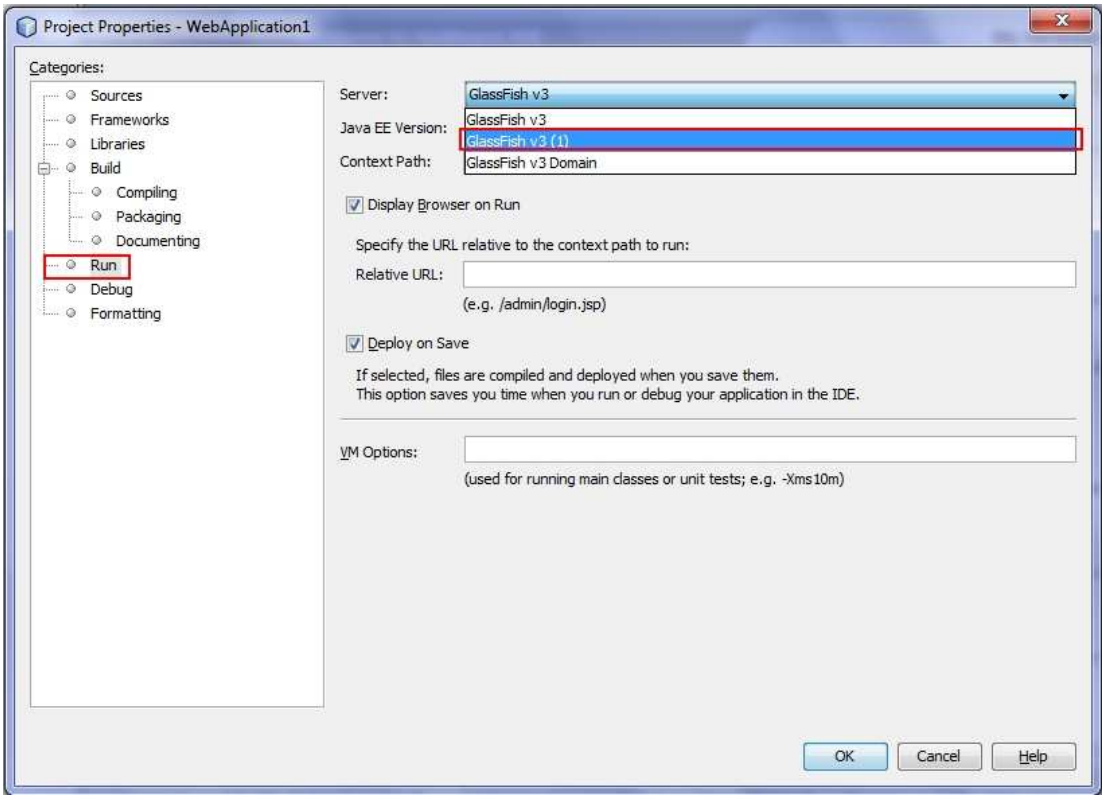


En regardant dans la propriété des serveurs, on remarque que le nouveau domaine n'utilise plus le port 8080 :

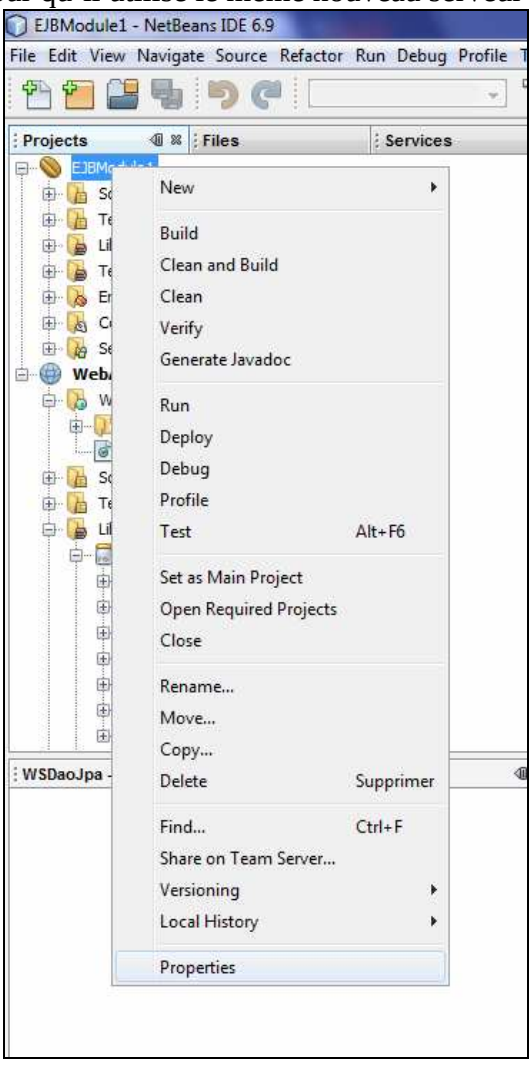


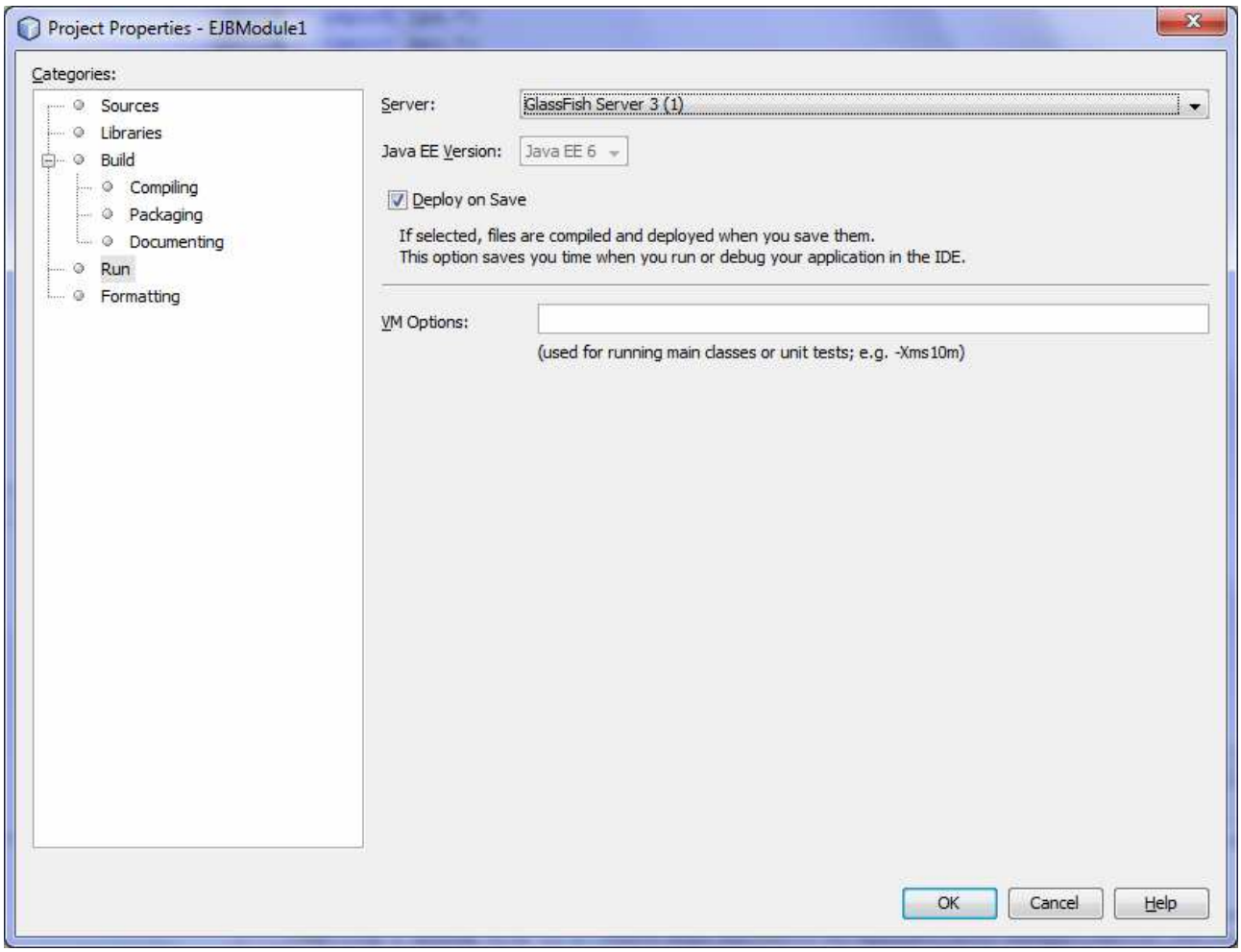
Il faut alors configurer le web service pour qu'il utilise le nouveau serveur (et pareillement pour les autre applications si besoin).


Proj...
Files
Services
Start P
EJBModule1
EnterpriseApplication3
ODBC
New
Build
Clean and Build
Clean
Generate Javadoc
Run
Deploy
Debug
Profile
Test RESTful Web Services
Test Alt+F6
Set as Main Project
Open Required Projects
Close
Rename...
Move...
Copy...
Delete
Find...
Share on Kenai...
Versioning
Local History
Properties



Ensuite reconfigurer l'EJB pour qu'il utilise le même nouveau serveur glassfish.





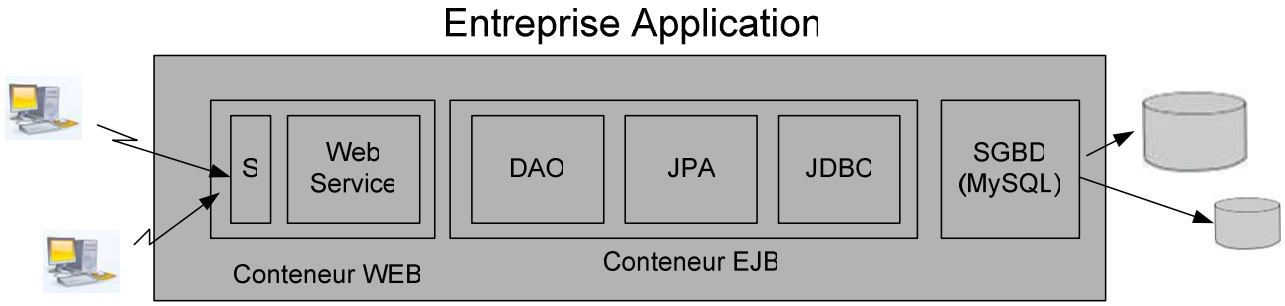
Enfin, redéployer le web service.
A moins que le nouveau port soit aussi utilisé, normalement, vous n’avez plus de problème de port.

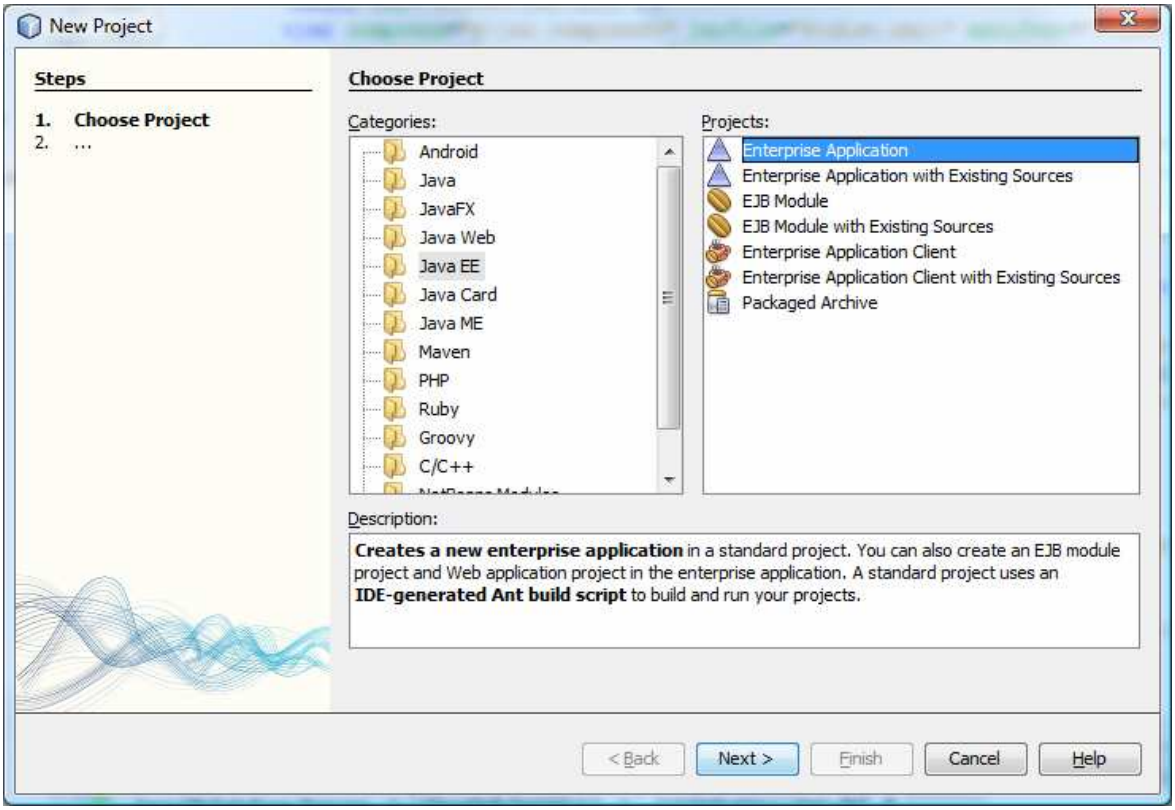
Partie 4. Création d'une EntrepriseApplication

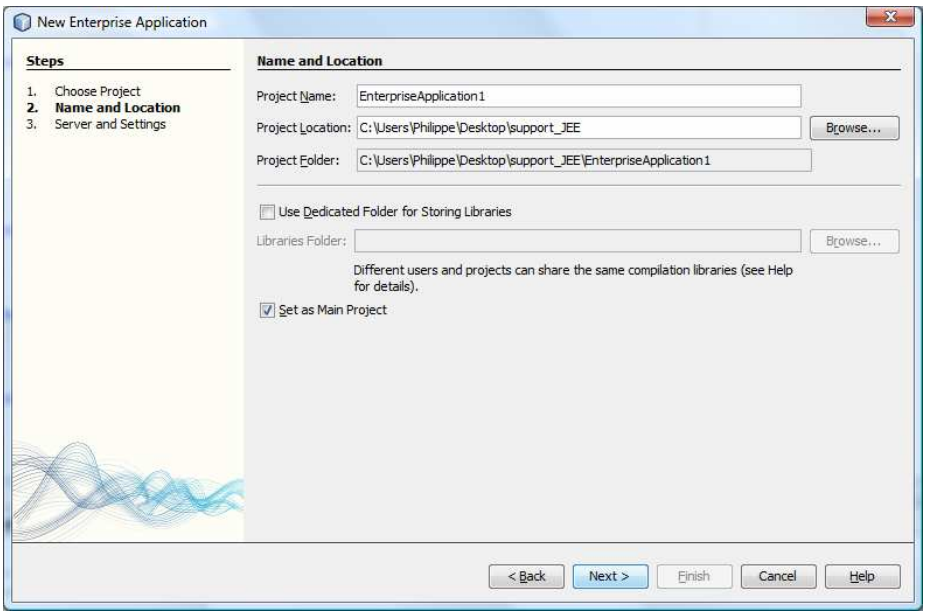
4.1. Création du projet

Un projet Entreprise Application rassemble dans un projet unique :

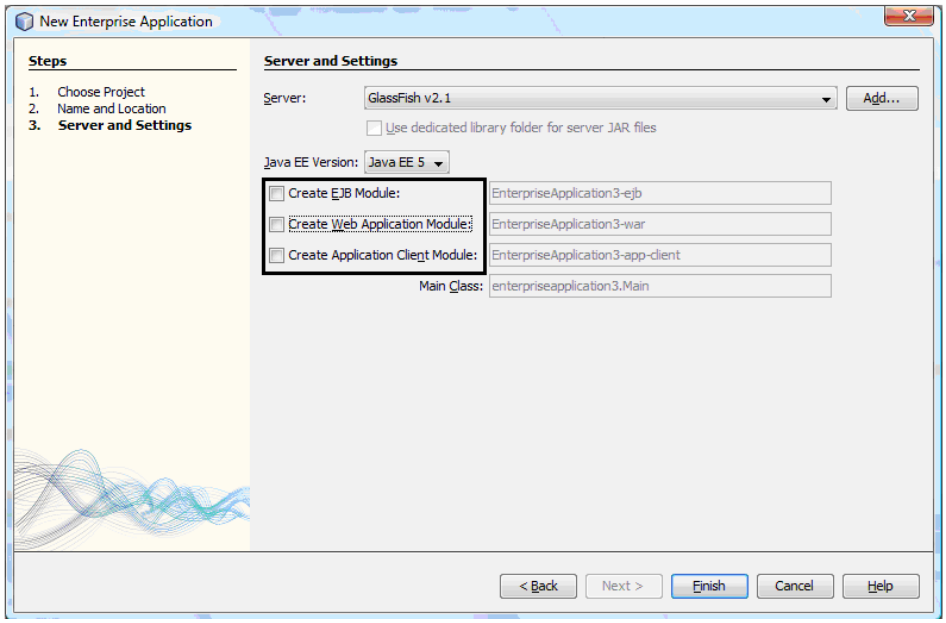
- l'EJB ;
- le Web Service.





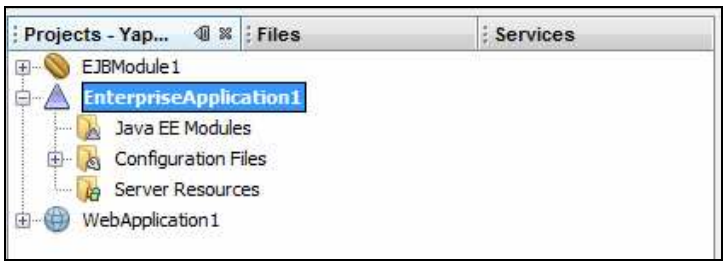


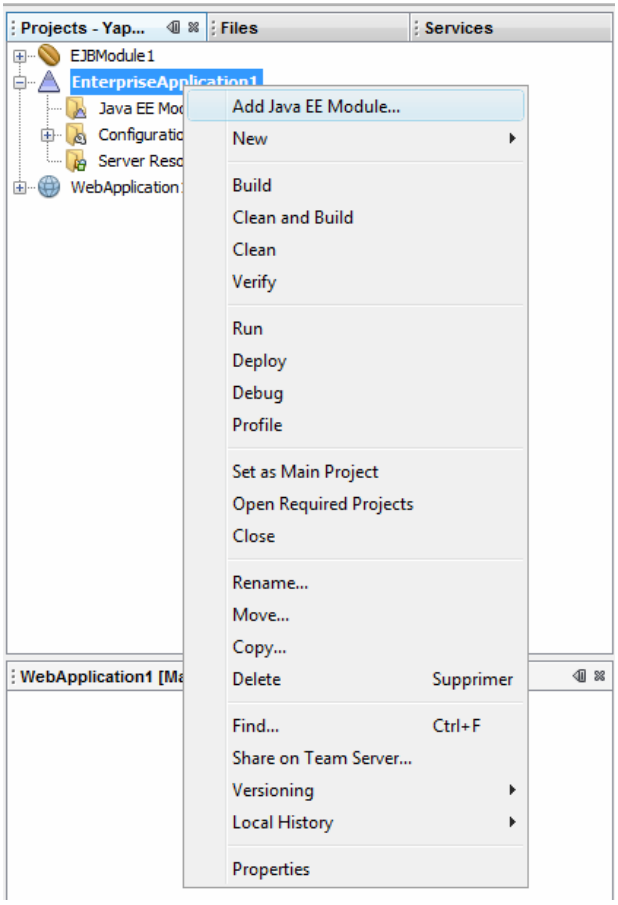
Attention à décocher les deux cases comme indiqué ci-dessous :

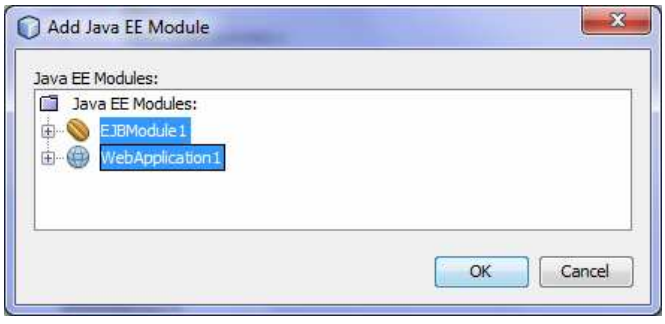


L'EJB et la WebApplication que nous venons de créer doivent être inclus dans l'application entreprise que nous venons de créer. Un clic droit sur le package « Java EE Modules » fait apparaître un menu contextuel qui recense les projets actuellement connus de NetBeans et référencés dans la partie projet.

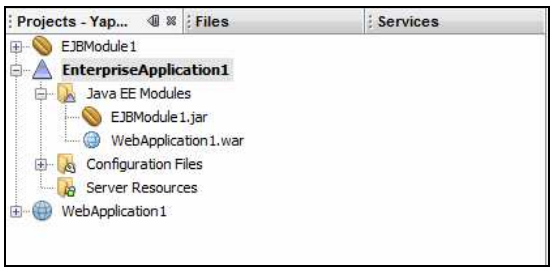
Vérifier que vous avez sélectionné le même serveur glassfish pour la web application l'EJB, et l''Entreprise application.





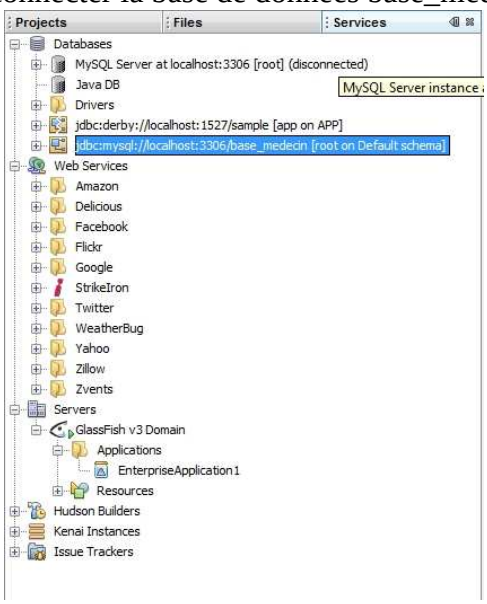


Une fois inclus dans le projet **EntrepriseApplication**, on doit obtenir ceci :

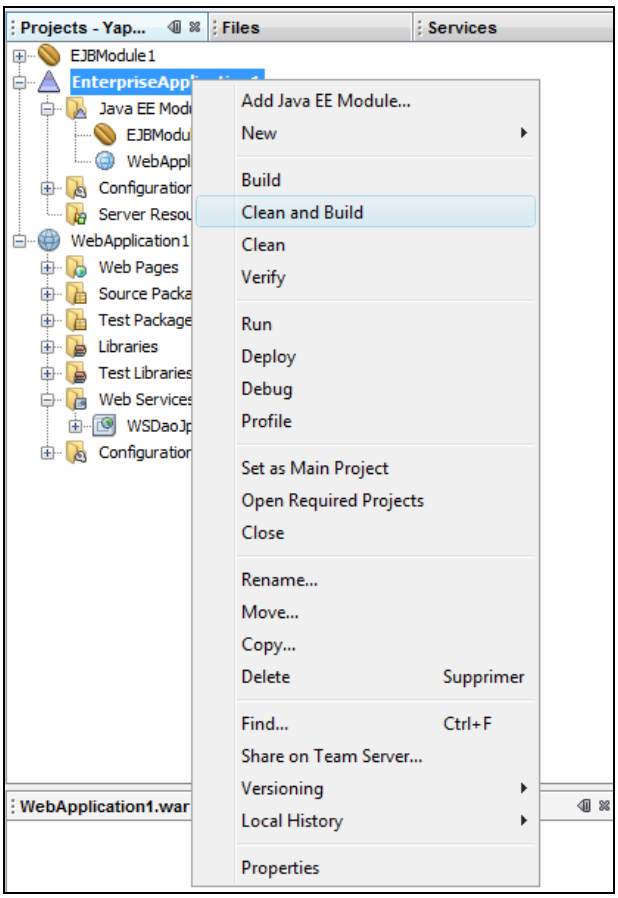


4.2. Tester le web service (again)

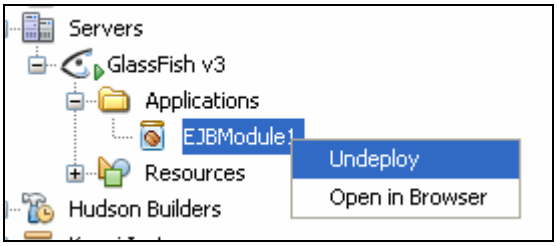
Penser à arrêter Glassfish et à connecter la base de données base_medecin avant de poursuivre.

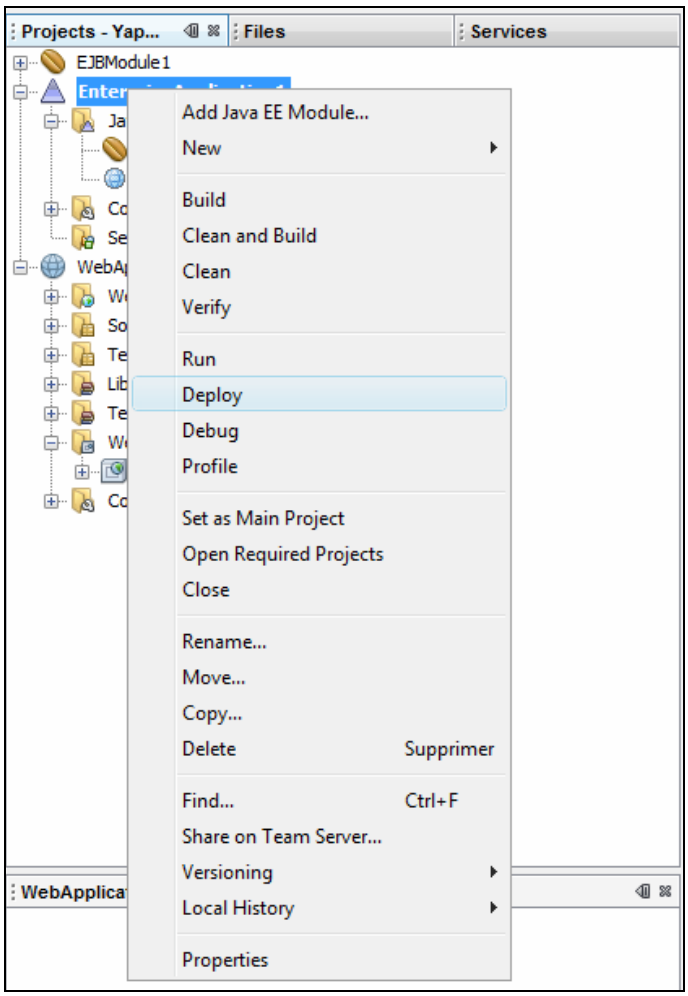


Dans un **premier temps**, il faut compiler le projet EntrepriseApplication1.

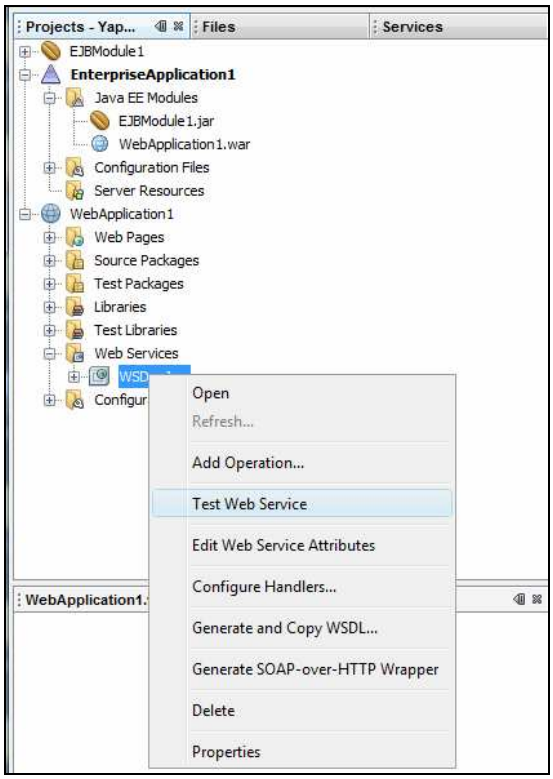


Dans un **deuxième temps**, il faut déployer le projet EntrepriseApplication1. Avant de déployer l'application assurez-vous d'avoir enlevé l'EJB déployé et/ou le Webservice afin d'éviter des conflits (Services→Servers→Glassfish→Application→EJB→Clic droit→Undeploy).

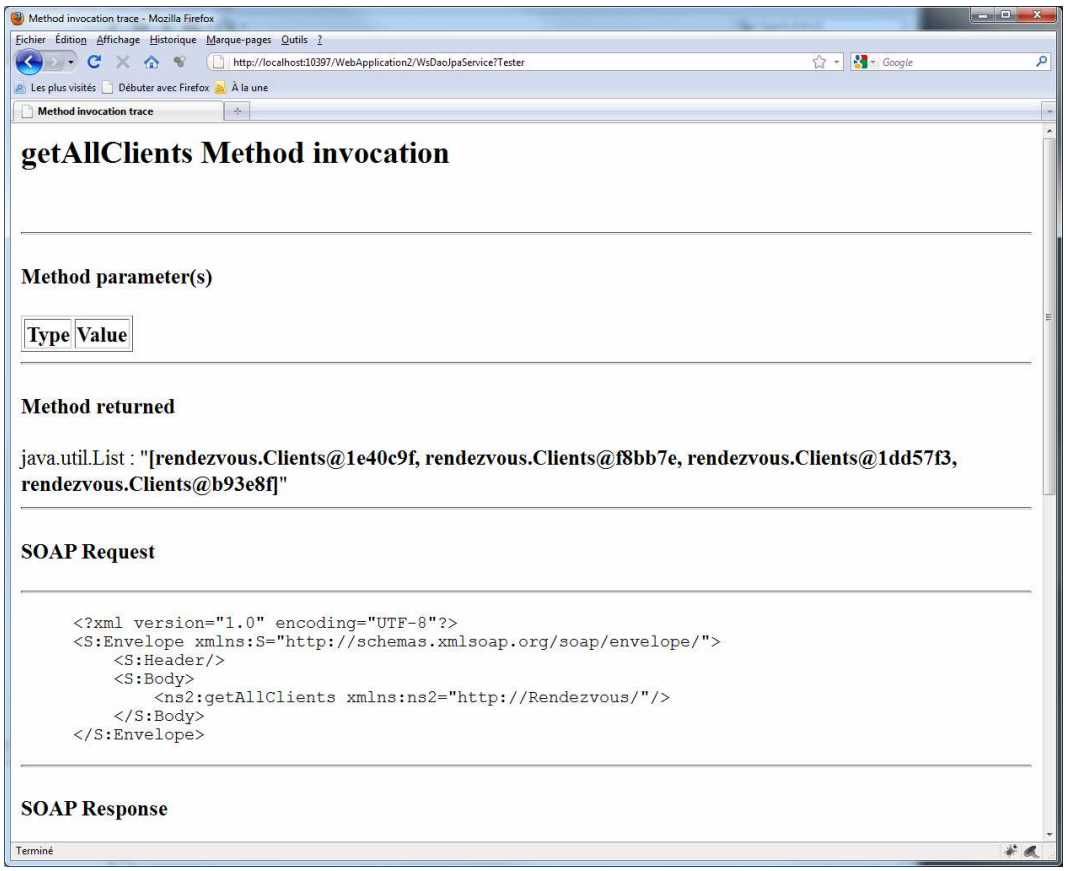




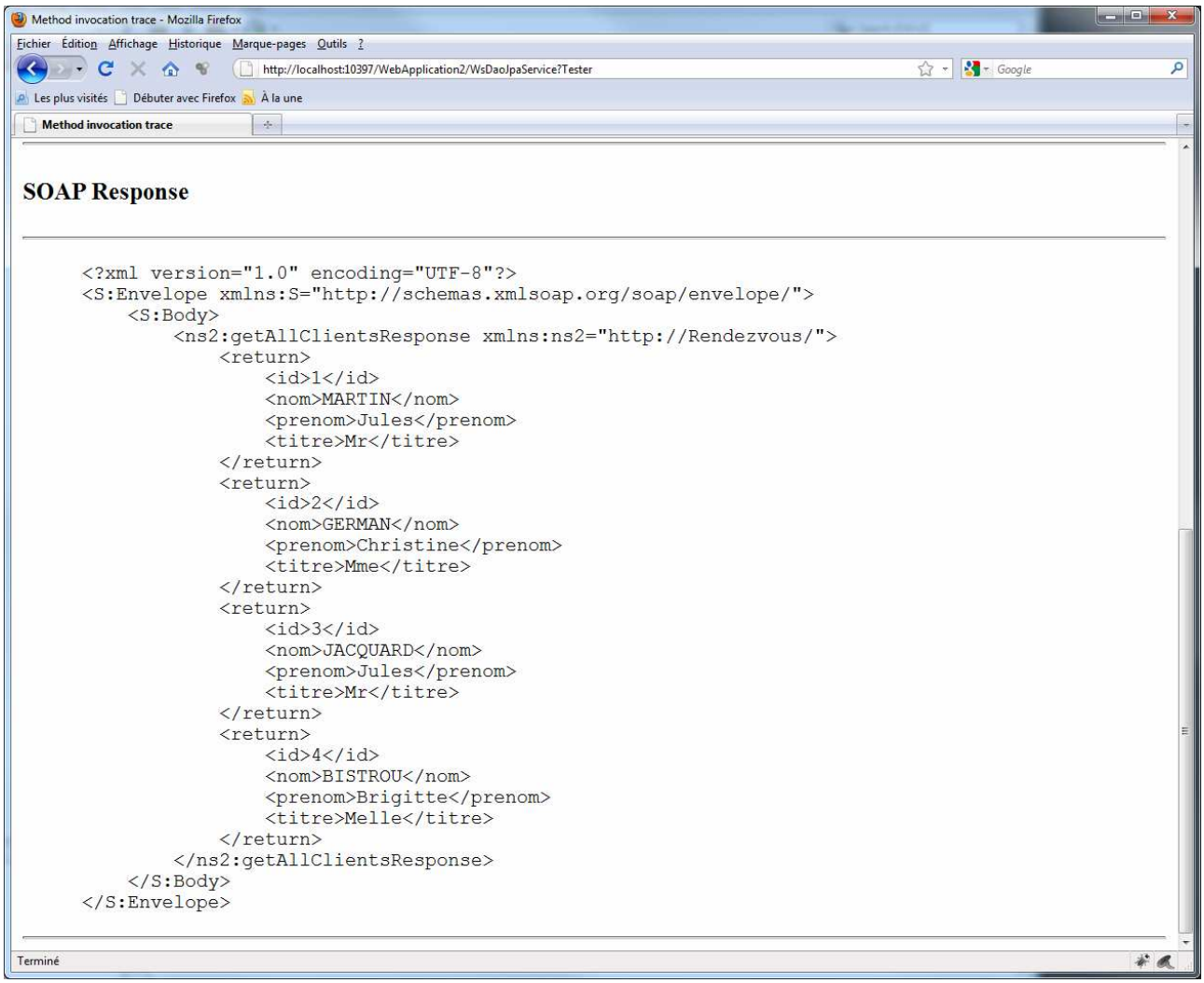
Finalement, faire un clic droit sur **WsDaoJpa** dans le projet WebApplication1. Choisir **Test WebService.**



Ceci permet de tester les web services directement à partir d'un navigateur !



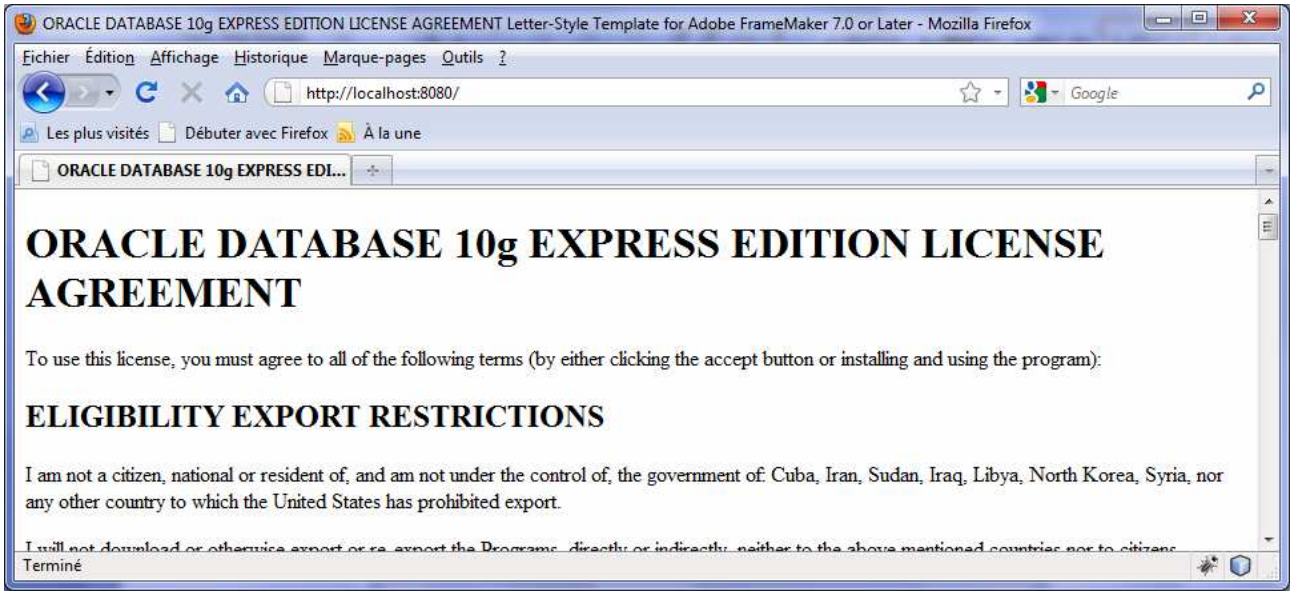
Avec la réponse SOAP :



Il peut arriver que cette partie ne fonctionne pas soit parce que vous n'avez pas déployé correctement, soit parce que le port est déjà utilisé par une autre application. Dans ces cas, vous aurez le message suivant

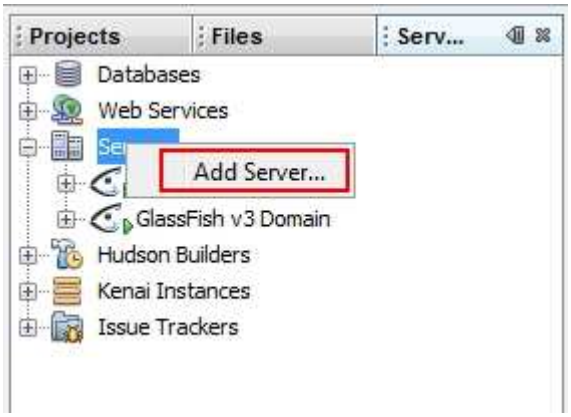
Si c'est un problème de déploiement, il faut vérifier tous les paramètres précédemment entrés. Si c'est un problème de port, il suffit de changer le port de glassfish.
Pour s'assurer que c'est bien un problème de port, on va entrer « localhost:8080 » (cf. message d'erreur) dans un navigateur.

Par exemple, nous avons obtenu ceci sur une machine de test :

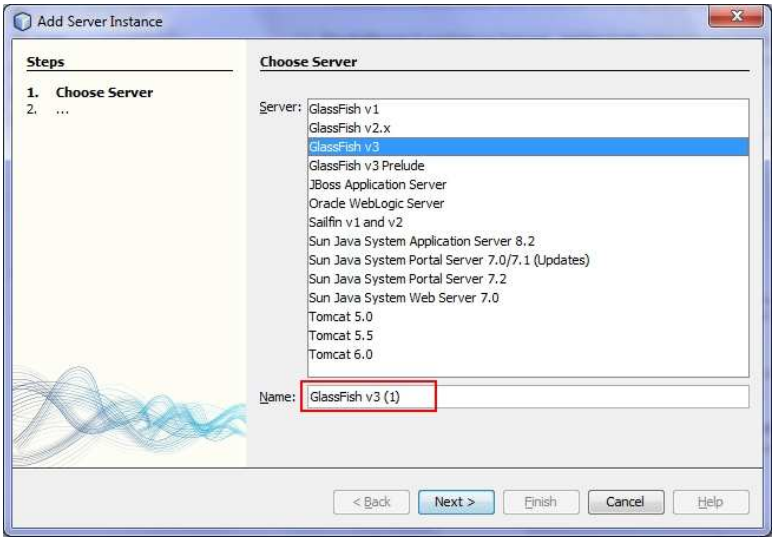


Cela indique que le port est occupé par oracle. Il faut donc dire à Glassfish d'utiliser un autre port.

Ajouter un nouveau serveur :

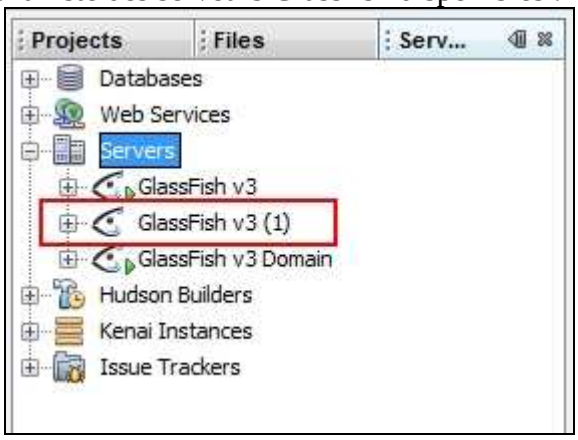


Et entrer un nom du nouveau domaine :

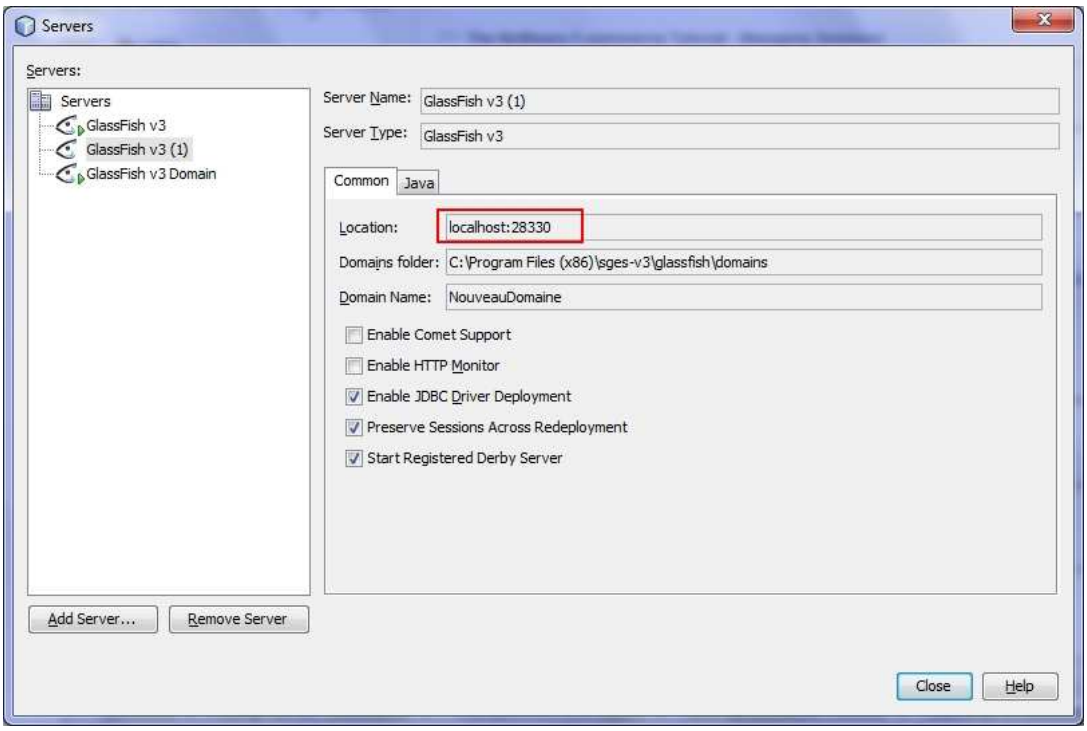


Puis valider les options par défaut.

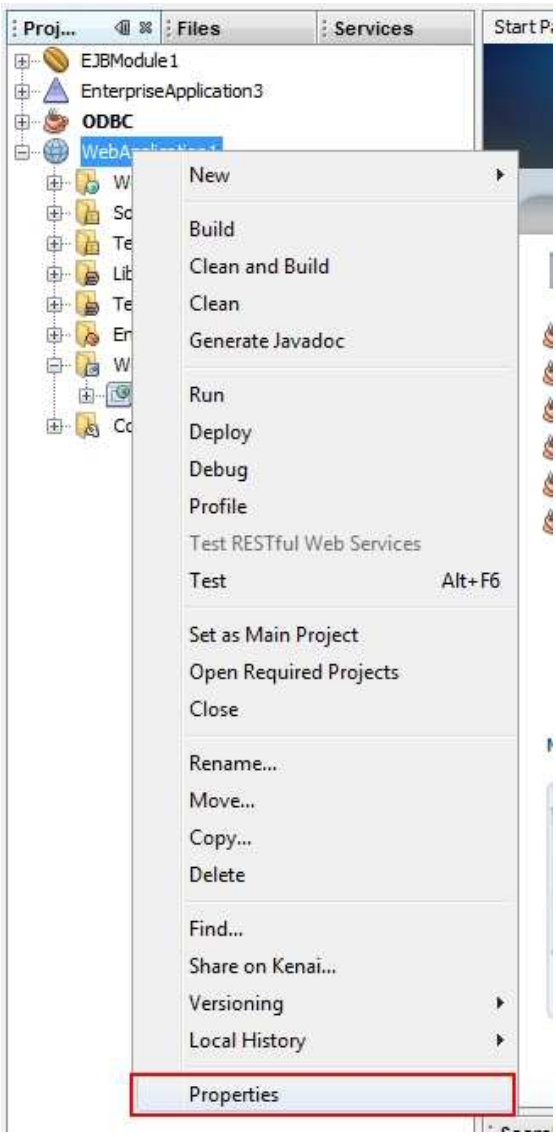
On retrouve ce serveur dans la liste des serveurs Glassfish disponibles :

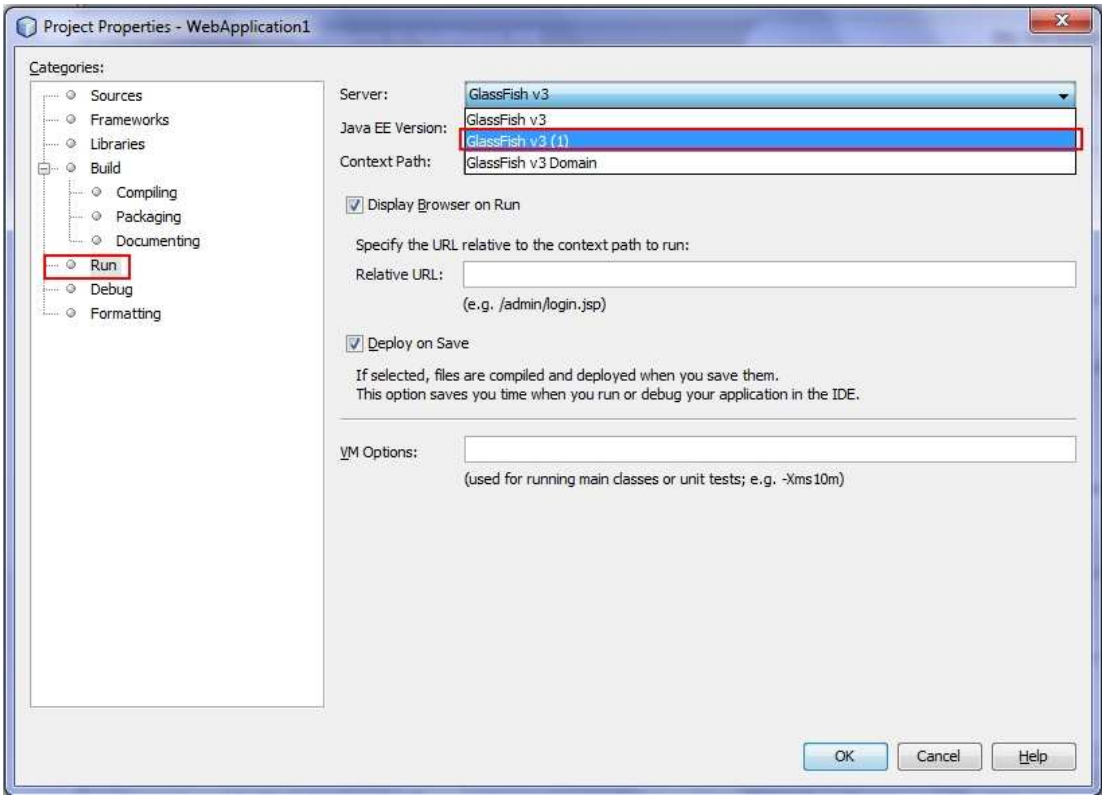


En regardant dans la propriété des serveurs, on remarque que le nouveau domaine n'utilise plus le port 8080 :

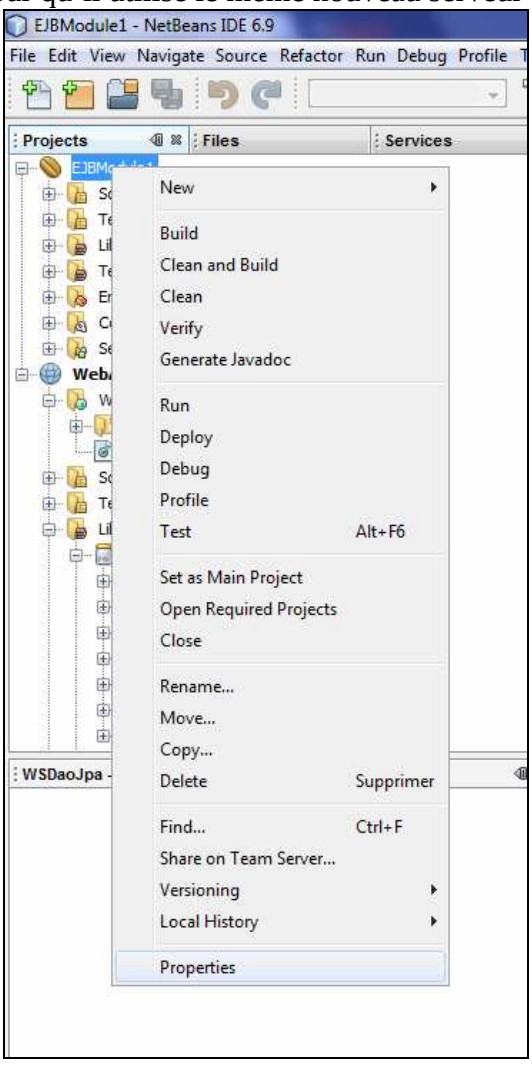


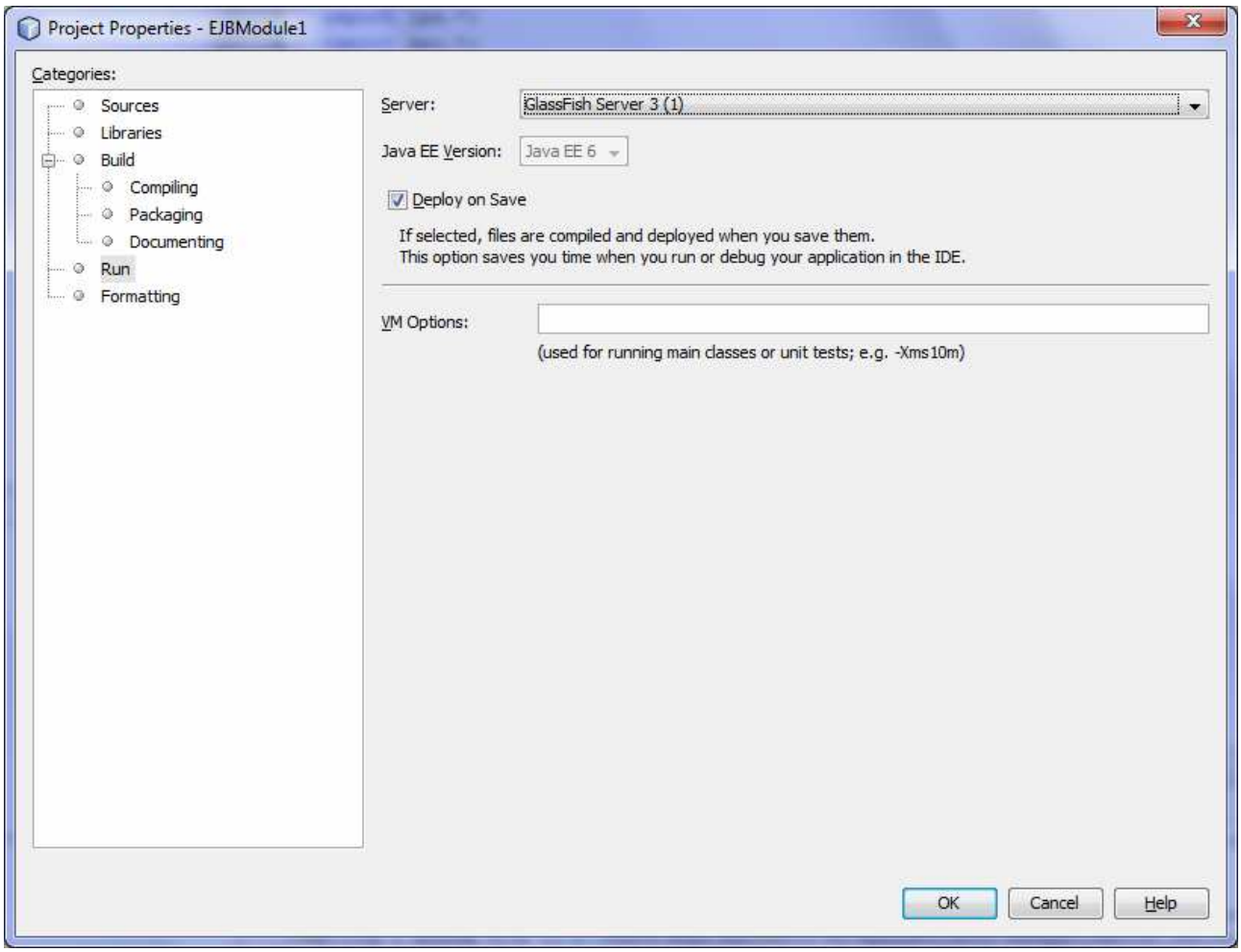
Il faut alors configurer le web service pour qu'il utilise le nouveau serveur (et pareillement pour les autre applications si besoin).


Proj...
Files
Services
Start P
EJBModule1
EnterpriseApplication3
ODBC
New
Build
Clean and Build
Clean
Generate Javadoc
Run
Deploy
Debug
Profile
Test RESTful Web Services
Test Alt+F6
Set as Main Project
Open Required Projects
Close
Rename...
Move...
Copy...
Delete
Find...
Share on Kenai...
Versioning
Local History
Properties



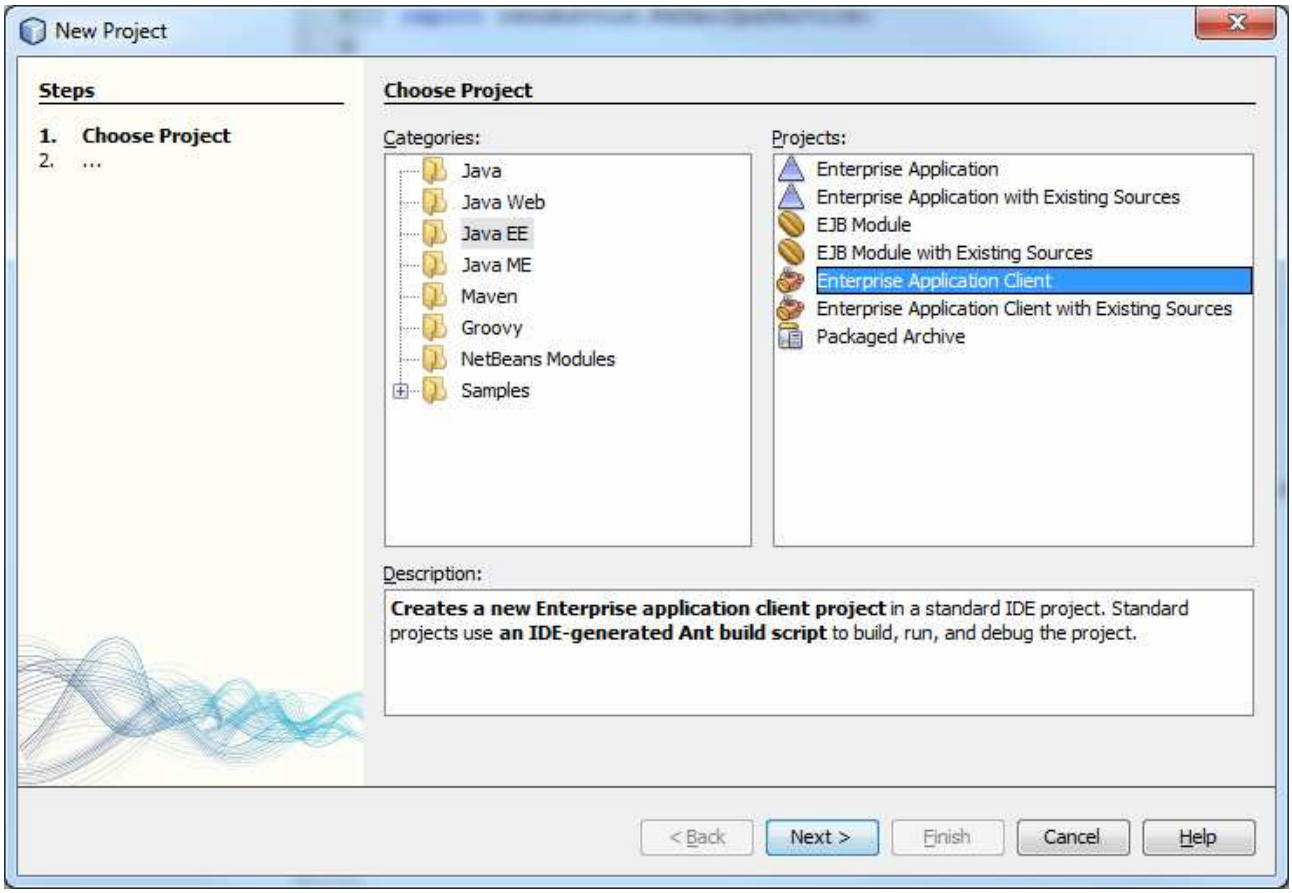
Ensuite reconfigurer l'EJB pour qu'il utilise le même nouveau serveur glassfish.

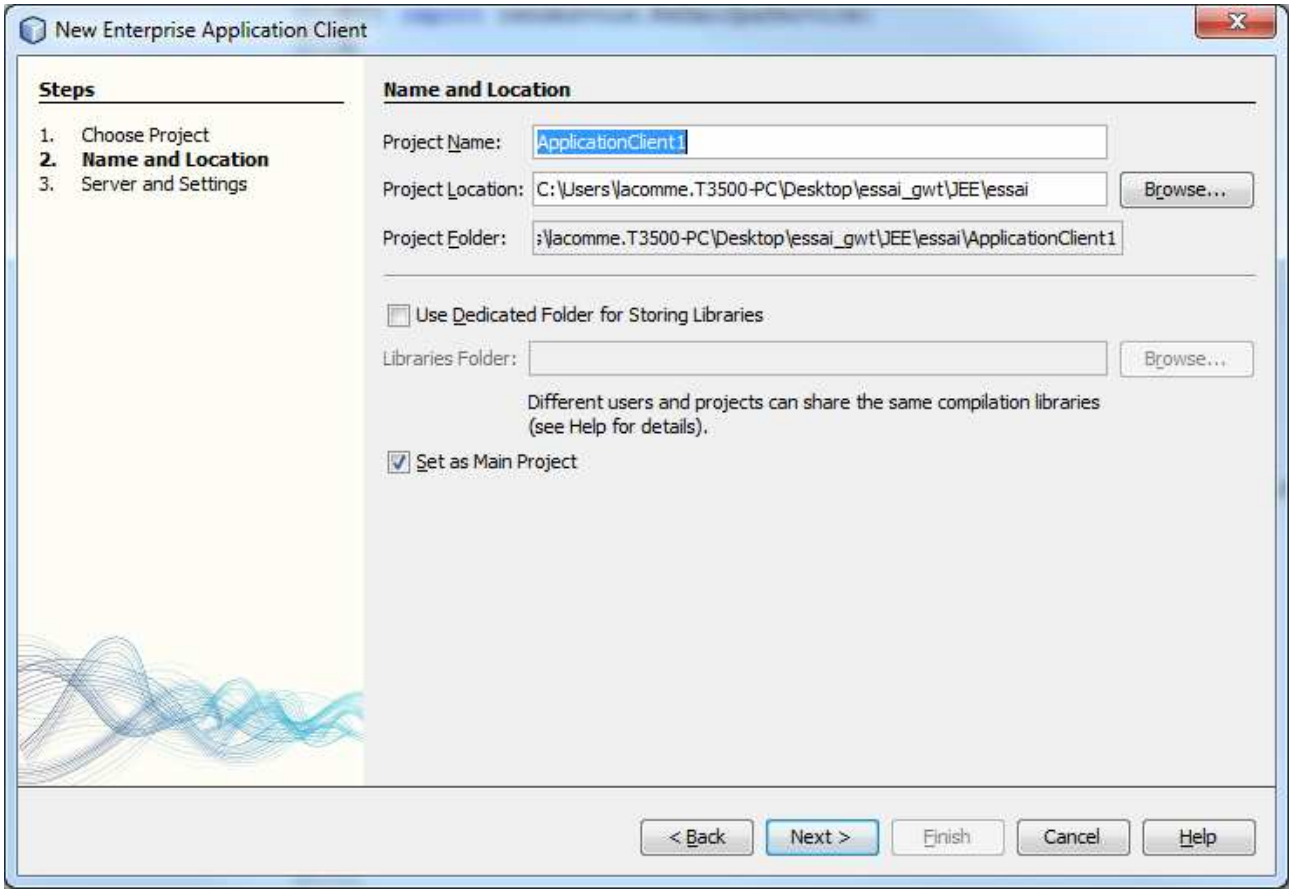


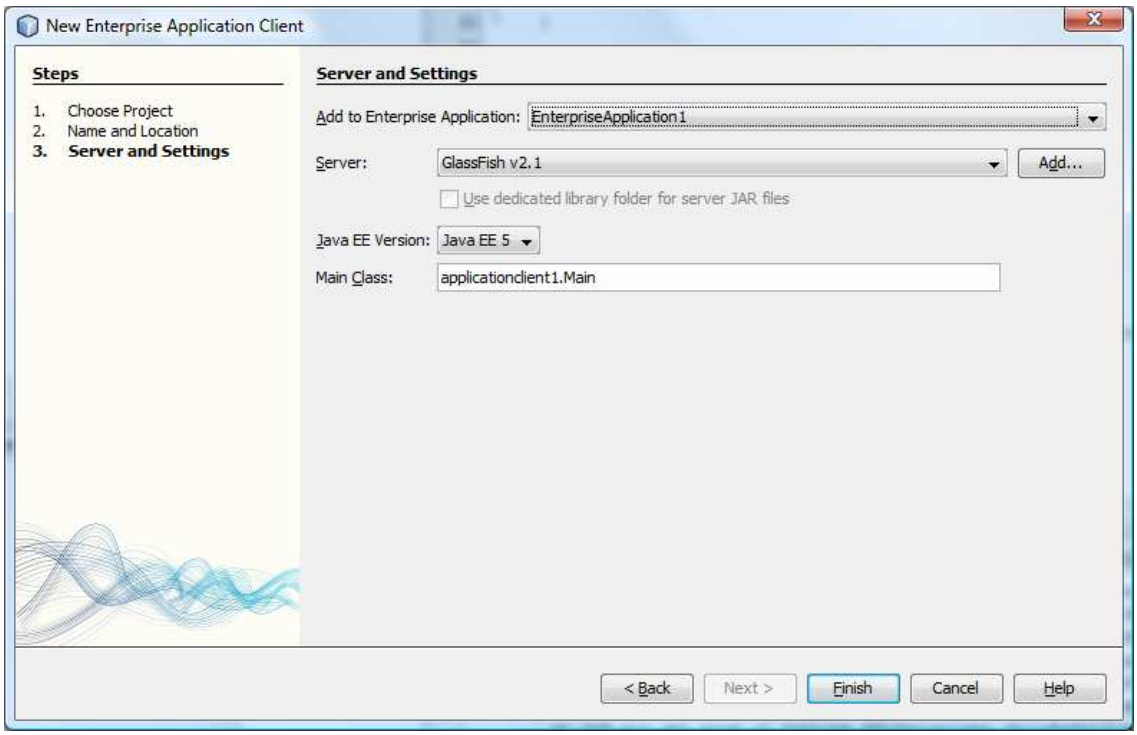


Enfin, redéployer le web service.
A moins que le nouveau port soit aussi utilisé, normalement, vous n’avez plus de problème de port.

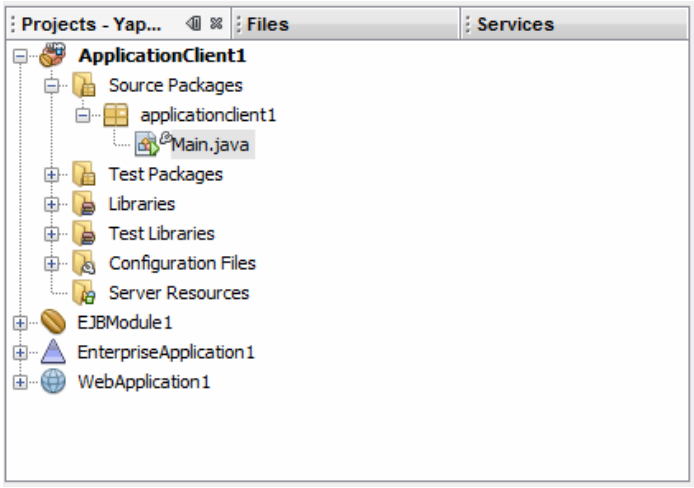
Partie 5. Création d'un client



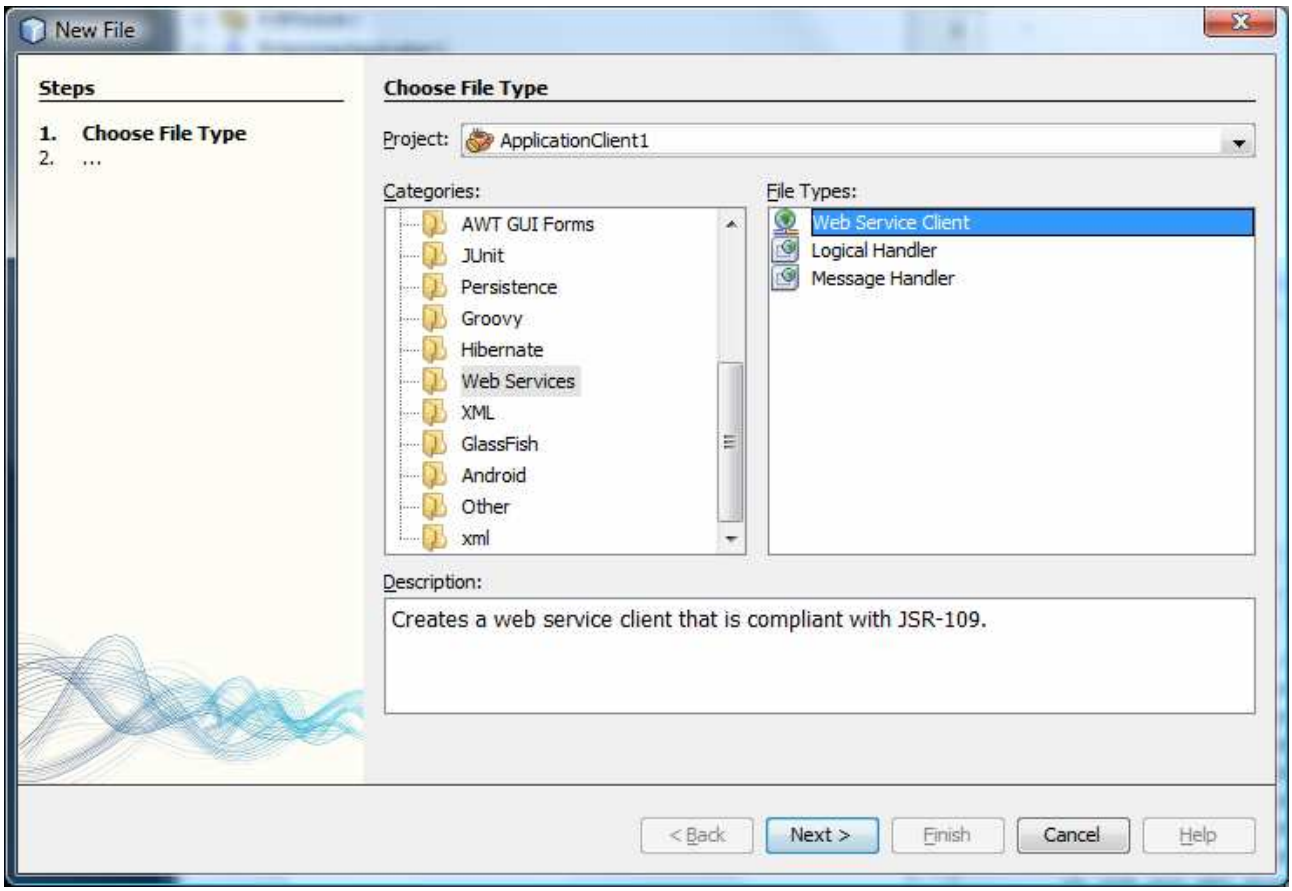




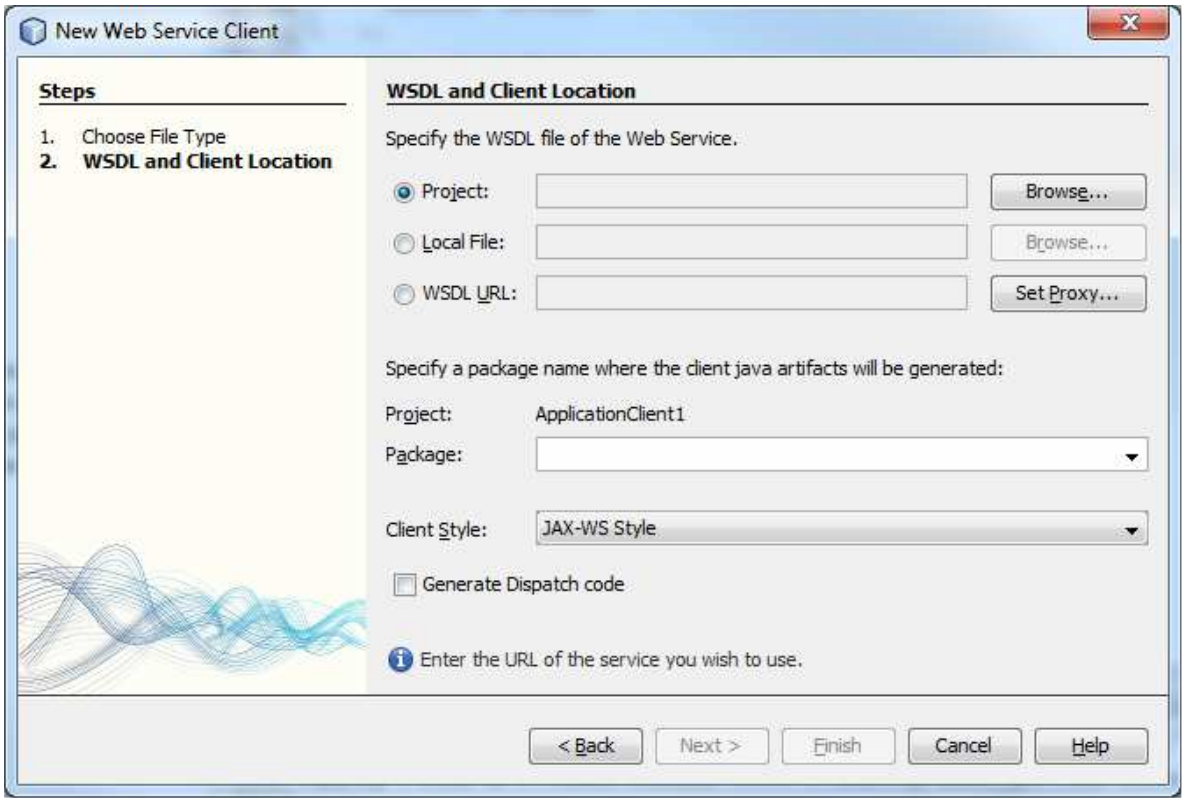
Ce qui donne au final :



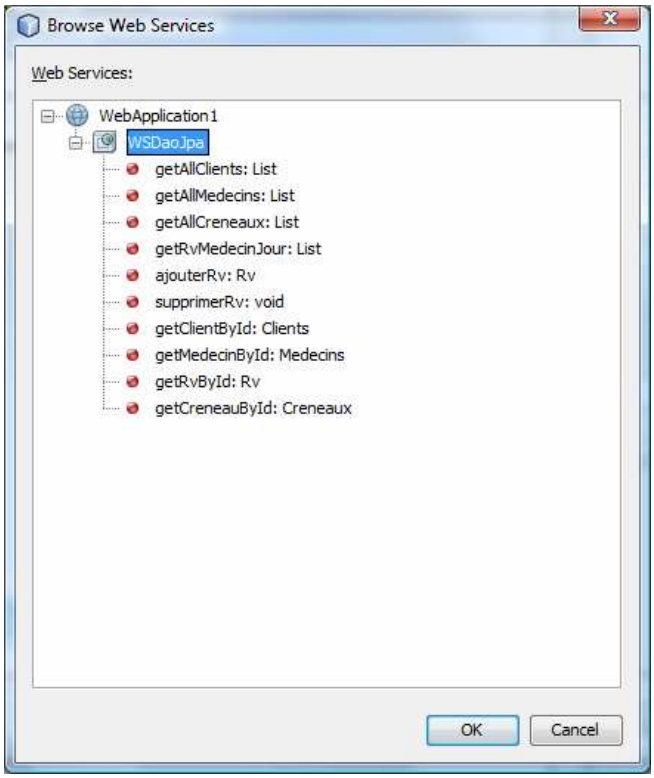
Faire ensuite new Web Service Client.



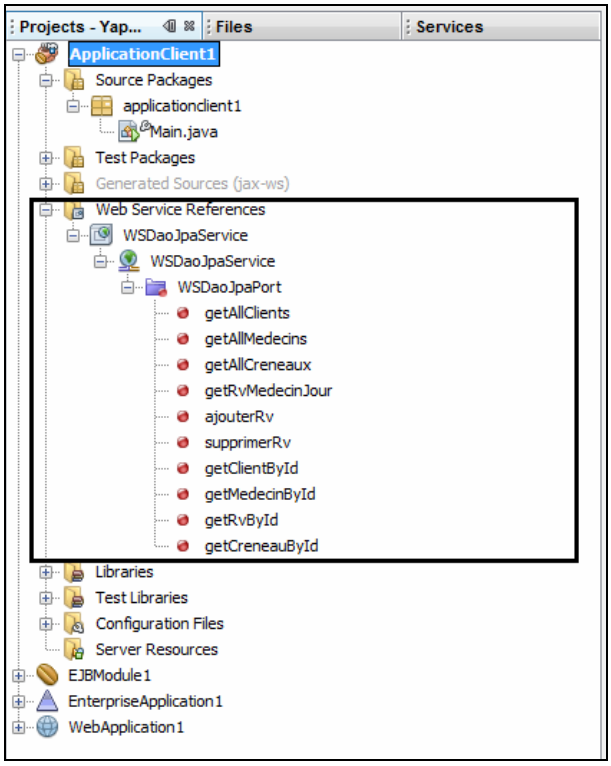
Cliquer sur « Broswer ».



Faire ensuite : Browse et choisir WebApplication1 puis WSDaoJpa.



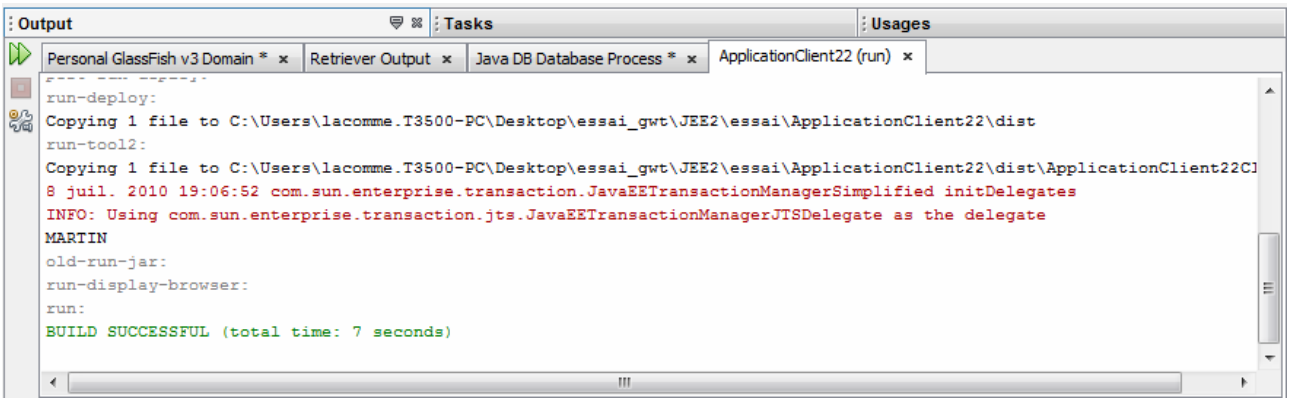
Ceci donne finalement un projet de la forme :



Le fichier main.java peut ensuite être modifié comme suit :

```
package applicationclient1;

import rendezvous.WSDaoJpaService;

public class Main {

    public static void main(String[] args) {
       WSDaoJpaService ws = new WSDaoJpaService ();
       System.out.println(ws.getWSDaoJpaPort().getAllClients().get(0).getNom());
    }

}
```

Ceci donne :

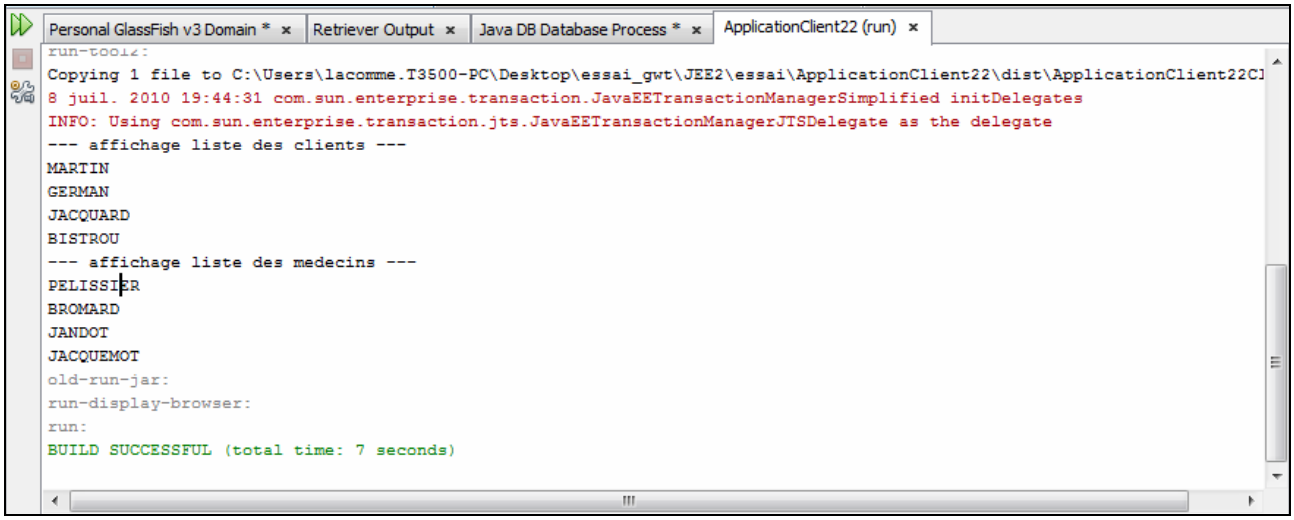


On peut avoir une version un peu améliorée :

```
package applicationclient1;

import java.util.*;
import rendezvous.* ;
import rendezvous.WSDaoJpaService;

public class Main {
    public static void main(String[] args) {
       WSDaoJpaService  ws = new WSDaoJpaService ();
       System.out.println("--- affichage liste des clients ---");
       List<Clients> myArr = new ArrayList<Clients>();
       myArr = ws.getWSDaoJpaPort().getAllClients();

       Iterator myArr_it = myArr.iterator() ;
       while(myArr_it.hasNext())
       {
        Clients ClientsCourant = (Clients) myArr_it.next() ;
        System.out.println(ClientsCourant.getNom());
       }

       System.out.println("--- affichage liste des medecins ---");
       List<Medecins> myArr_m = new ArrayList<Medecins>();
       myArr_m = ws.getWSDaoJpaPort().getAllMedecins();

       int Taille = myArr.size() ;
       for (int i=0 ;i<Taille ; ++i)
       {
```

```
        Medecins MedecinCourant = myArr_m.get(i) ;
        System.out.println(MedecinCourant.getNom());
       }
    }
}
```

Ce qui donne :



---------- FIN ----------