Questions Systèmes Distribués

Partie 1 : EJB et API de persistence

1) Quels sont les différents rôles que peut jouer une personne sur la plateforme JEE

Les modules réutilisables permettent de diviser le processus développement et de déploiement en plusieurs rôles distinct afin de permettre à différentes personnes ou sociétés de réaliser un des modules. Les deux premier rôles impliquent d'acheter et d'installer le produit et les outils de Java EE. Une fois acheté et installé, les composants de Java EE peuvent être développés par des fournisseurs de composants d'application, être assemblés par des assembleurs d'applications, et être déployés par des « déployeurs » d'applications. Chacun de ces rôles peuvent être exécutés par des personnes différentes.

Java EE Product Provider : il s'agit de l'entité (personne, compagnie, ...) qui va analyser, concevoir et mettre en vente des API de la plateforme JEE. Typiquement, il s'agit d'un fournisseur de serveur d'application programmée en respectant les spécifications JEE.

Tool Provider : il s'agit de la personne ou de la société qui fournit les outils pour développer, assembler, emballer les produits proposés par les fournisseurs de composant, les assembleurs et les exploitants.

Application Component Provider : il s'agit de la personne ou la société qui fournira les composants Web, les applets, les « enterprise beans »et des applications client pour être utilisé dans des applications JEE.
Le développeur des « enterprise beans » réalise les tâches suivantes pour fournir un EJB JAR qui contient un ou plusieurs « enterprise beans » : il écrit et compile le code source, spécifie le descripteur de déploiement et empaquette les fichiers .class et le descripteur de déploiement dans un EJB JAR.
Le développeur de composant web réalise les tâches suivantes pour fournir un fichier WAR qui contient un ou plusieurs composants web : il écrit et compile le code source de la servlet, il écrit les JSP, JavaServer faces et page HTML, spécifie le descripteur de déploiement et empaquette les fichiers .class, .jsp et .html et le descripteur de déploiement dans un fichier WAR.
Le développeur de l'application client réalise les tâches suivantes pour fournir un JAR qui contient l'application cliente : il écrit et compile le code source, spécifie le descripteur de déploiement pour le fichier, et empaquette les fichiers .class et le descripteur de déploiement dans un JAR.

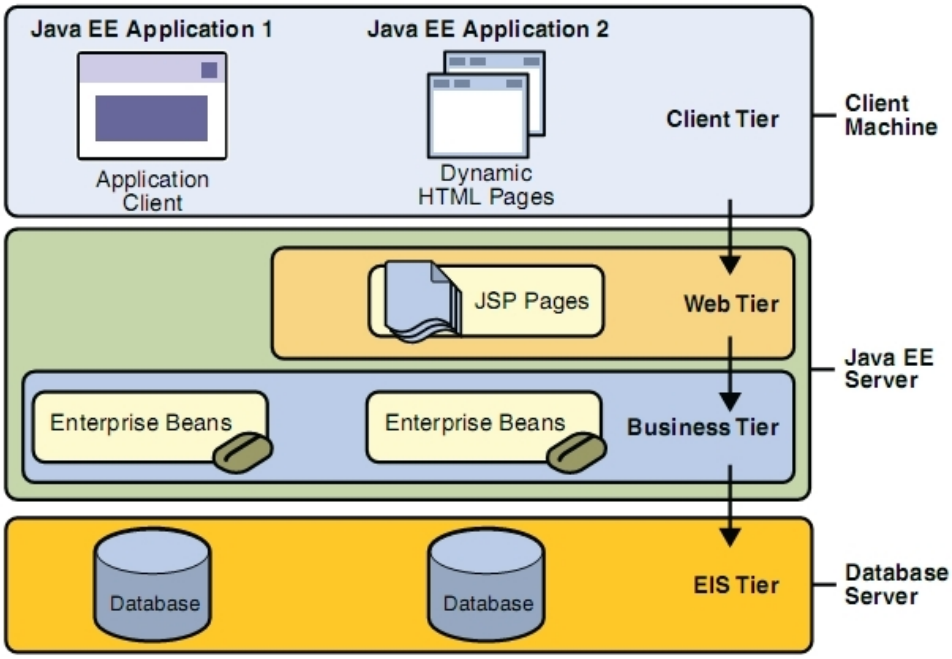
Application Assembler : il s'agit de la personne ou la société qui reçoit les modules d'application à partir des fournisseurs de composants et les assemble dans un fichier EAR basé sur les spécifications JEE. Cette personne sera chargée de construire un fichier de déploiement afin que l'application finale soit correctement implantée.
Un développeur d'application réalise les tâches suivantes pour fournir un EAR contenant l'application JEE : il assemble les EJB JAR et WAR créé dans les phases précédentes en une application Java EE (EAR), il spécifie le descripteur de déploiement pour l'application Java EE et vérifie que le contenu de l'EAR est bien formé et satisfait la spécification de Java EE.

Application Deployer and Administrator : c'est la personne ou la société qui va administrer l'environnement d'exécution où les applications JEE tournent. Il est chargé de configurer le système en fonction des besoins du client, de gérer la sécurité ainsi que les droits d'accès aux bases de données éventuelles.
La personne qui déploie ou administre l'application réalise les tâches suivantes pour installer et configurer l'application Java EE : il ajoute l'application Java EE (EAR) créée dans les phases précédentes au serveur Java EE, configures l'application Java EE pour un environnement opérationnel en modifiant le descripteur de déploiement de l'application, vérifie que le contenu de l'archive EAR est bien formé et satisfait aux spécification Java EE et déploie le EAR de l'application Java EE sur le serveur Java EE.

2) Quelle est la place(la fonction remplie) des EJB dans un système multi-tiers ? Expliquez à l'aide d'un schéma.

Le modèle n-tier est un modèle logique d'architecture applicative qui vise à séparer nettement trois couches logicielles au sein d'une même application et à présenter l'application comme un empilement de ces couches :

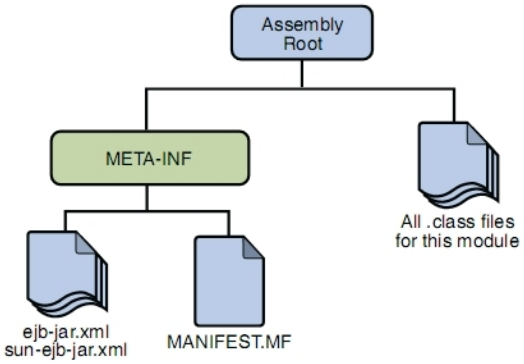
- présentation des données
- traitement métier des données
- accès aux données persistantes



Les couches communiquent entre elles au travers d'un « modèle d'échange », et chacune d'entre elles propose un ensemble de services rendus.
Les services d'une couche sont mis à disposition de la couche supérieure.

Le « tier » business (et donc les EJB), est invoqué par le « tier » présention et implémente la logique métier de l'application. Le « tier » persistant est utilisé par le « tier » business pour accéder à des données externes comme des bases de données ou d'autres applications.
Les EJB ont donc pour but l'encapsulation de la logique métier c'est à dire la transaction, la sécurité, l'évolutivité, la concurrence, les communications, la gestion des ressources, la persistances, la gestion des erreurs, l'indépendance de l'environnement d'exploitation.

3) Décrire la structure et le contenu d' une archive EJB.



Pour permettre le déploiement d'un EJB, il faut définir un fichier DD (deployement descriptor) qui contient des informations sur le bean. Ce fichier au format XML permet de donner au conteneur d'EJB des caractéristiques du bean.

Un EJB doit être déployé sous forme d'une archive jar qui doit contenir un fichier qui est le descripteur de déploiement et toutes les classes qui composent chaque EJB (interfaces home et remote, les classes qui implémentent ces interfaces et toutes les autres classes nécessaires aux EJB).

Une archive ne doit contenir qu'un seul descripteur de déploiement pour tous les EJB de l'archive. Ce fichier au format XML doit obligatoirement être nommé ejb-jar.xml.

L'archive doit contenir un répertoire META-INF (attention au respect de la casse) qui contiendra lui même le descripteur de déploiement.

Le reste de l'archive doit contenir les fichiers .class avec toute l'arborescence des répertoires des packages.

4) Quelles sont les différences entre interface Local et Remote ?

En local, le client appelant est exécuté dans la même machine virtuelle Java (JVM) que celle de l'EJB (à distance, dans la JVM du client).

L'utilisation de l'interface distante implique la **sérialisation** et RMI (Remote Method invocation), donc le passage d 'objets au travers du réseau. L'utilisation de l'interface locale est **directe**, donc beaucoup plus rapide. Limiter le nombre d'appels distants est un facteur important pour la performance de votre système.

Lors d'un appel distant, les paramètres sont passés par **valeur**, la référence n'ayant aucun sens dans ce cas. Lors d'un appel local, les paramètres sont passés par **référence**. Des comportements différents existent donc entre les deux appels.

Dans le cas d'un interface local, comme on est sur la même jvm, on ne peut plus utiliser de **load/balancing** et autres mécanismes répartis.

5) Citez et expliquez deux méthodes pour qu'un client obtienne l'interface business d'un EJB session. Peut-on utiliser ces méthodes dans tous les cas ? Pourquoi ?

- En récupérant le contexte initial du serveur. En accédant à un contexte JNDI, on peut récupérer une instance de contexte dans lequel on peut avoir les informations concernant les interfaces locales et remote afin d'en invoquer les méthodes.
- En utilisant des annotations spécifiques, on peut récupérer les informations qui nous intéressent. Ces annotations vont avoir pour effet qu'à l'exécution, une instance de l'objet remote concerné sera créée depuis le container du serveur.

Ces 2 méthodes sont utilisables tout le temps à partir du moment où on travaille avec des EJB v3.0, car les annotations ne fonctionnent pas sur les versions précédentes.

6) Quels sont les class files qu'un Bean provider doit produire en EJB 3.0? En EJB 2.X ?

EJB 3.0 :

- écrire les interfaces Remote et locale de l'ejb

```
@Local
public interface HelloWorldLocal
{
	// types serializable ... RMI
	String sayHello();
}
@Remote
public interface HelloWorldRemote
{
	String sayHello();
}
```

- écrire la classe d'implémentation

```
import javax.ejb.Stateless;
@Stateless(mappedName="helloWorldBeanJndi")
public class HelloWorldBean implements
HelloWorldRemote, HelloWorldLocal
{
	// un constructeur indispensable ...
	public HelloWorldBean() {}
	public String sayHello()
	{
		return "Hello World";
	}
}
```

- écrire le client

```
import javax.naming.*;
public class Main
{
	@EJB
	private static HelloWorldRemote helloWorldBean;
	public static void initRefBean(String beanName)
	{
		Context c = new InitialContext();
		// ! NamingException ...
		helloWorldBean =(HelloWorldRemote) c.lookup(beanName);
	}
}
```

- écrire le descripteur de déploiement

EJB 2.X :

- écrire l'interface RemoteHome et RemoteBusiness

```
package helloworldpackage;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
public interface HelloworldRemoteHome extends EJBHome
{
	HelloworldRemote create() throws CreateException,
	RemoteException;
}
```

```
public interface HelloworldRemoteBusiness
{
      String sayhello() throws java.rmi.RemoteException;
}

public interface HelloworldRemote extends EJBObject,
HelloworldRemoteBusiness{}
```

- écrire la classe d'implémentation

```
public class HelloworldBean
implements SessionBean, HelloworldRemoteBusiness
{
      private SessionContext context;
      public void setSessionContext(SessionContext aContext)
      {
            context = aContext;
      }
      public void ejbActivate() {
      public void ejbPassivate() { }
      public void ejbRemove() { }
      public void ejbCreate() { }
      public String sayhello() {return "Hello World";}
}
```

- écrire le client

```
Context c = new InitialContext();
Object objref = c.lookup("MyHelloWorld");
HelloworldRemoteHome rv =
     (HelloworldRemoteHome)
     PortableRemoteObject.narrow(objref,
     HelloworldRemoteHome.class);

HelloworldRemote v = rv.create();
rv.sayhello();
```

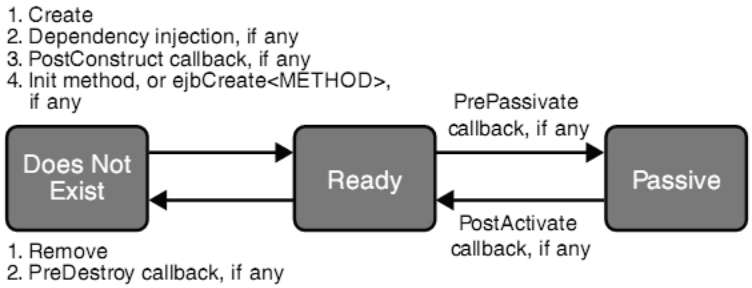
- écrire le descripteur de déploiement

```
<display-name>hel lo-ejb </ display-name>
   <enterprise-beans>
      <session>
         <display-name>helloworldSB</display-name>
         <ejb-name>HelloworldBean</ejb-name>
         <home>HelloworldRemoteHome</home>
         <remote>HelloworldRemote</remote>
         <ejb-class >HelloworldBean</ejb-class >
         <session-type>Stateless</session-type >
         <transaction-type>Container</transaction-type >
      </session>
   </enterprise-beans>
</ejb-jar>
```

7) Comparez à l'aide d'un schéma les EJB session Stateful des EJB session Stateless.

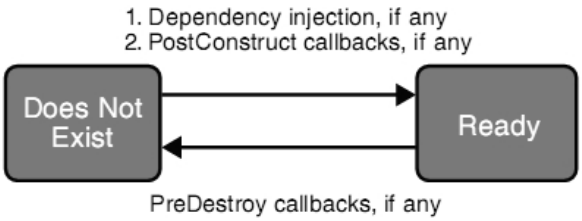
Un session bean avec état (statefull) :

- est une extension d'un client (sa durée de vie est celle du client);
- effectue une tâche pour un unique client en maintenant un état conversationnel entre deux appels de méthodes;
- a souvent besoin d'être initialisé;
- est souvent utilisé pour faire du suivi de session pour un client
- lorsqu'un client appelle l'EJB, une instance de ce dernier est créée, puis sert le client. Cette instance reste disponible pour les futurs appels de ce client uniquement. Cette instance sera détruite à la fin de la session (timeout ou appel à une méthode portant l'annotation @Remove)
- Si il y a trop d'instances d'un EJB en mémoire, ces dernières peuvent être sorties de la mémoire de travail. Elles passent ainsi en mode passif (= sauvées sur le disque et donc tous les attributs doivent être sérialisables => implémenter l'interface Serializable).



Un session bean sans état (stateless) :

- a un ensemble de méthodes qui utilisent leurs seuls paramètres pour produire un résultat (ils peuvent aussi utiliser leur SessionContexts);
- peut être utilisé par plusieurs clients
- lorsqu'un client appelle l'EJB, une instance de ce dernier sert le client puis retourne dans le pool d'EJB (cette dernière est donc prête à être réutilisée pour un autre client)
- sa durée de vie peut dépasser celle d'un client : un client peut conserver une connexion avec un objet EJB mais l'instance du bean peut servir plusieurs clients;
- peut avoir des variables d'instance par exemple pour déterminer le nombre de fois où il a été appelé ou pour le débogage (cet état n'est jamais visible pour le client car il ne sait jamais si il a affaire à la même instance);
- il consomme peut de ressource
- est spécialement pensé pour être robuste et fiable lorsqu'il y a beaucoup d'appels en concurrence



8) Citez et expliquez trois fonctions remplies par un container d'EJB.

Le conteneur offre un support d'exécution et plusieurs services à un EJB. Les services offerts par le conteneur d'EJB sont les suivants :

- **gestion du cycle de vie** : le conteneur gère le cycle de vie d'un EJB. Il offre aux beans des services tel que l'initialisation, la gestion de thread, l'allocation de ressource, et de destruction d'objets. Le conteneur gère également le pool de beans. Quand un bean présent en mémoire n'est plus utilisé, le conteneur se charge de l'y retirer, ainsi la mémoire peut être utilisée par d'autres ressources.
- **gestion de transaction** : une transaction est un groupe d'activités exécutées par une unité unique. Un EJB peut utiliser à la fois des transactions locales et distribuées par le biais du conteneur. Pour gérer les transactions, le conteneur utilise l'API de transaction Java (JAT).
- **gestion de sécurité** : la sécurité d'un EJB est assurée en utilisant les différentes entrées présentes dans le descripteur de fichier de déploiement.
- **Persistance** : le conteneur est responsable de l'enregistrement et de l'extraction des données persistantes. Il utilise le mécanisme de gestion de persistance pour sauver et restaurer une entité dans un milieu persistant comme par exemple une base de données.
- **services de désignation et d'enregistrement** (naming and registry services) : le conteneur EJB et le serveur fournissent un EJB ayant accès aux services nommés.
- **support pour client multiple** : le conteneur EJB fournit un support pour des clients de type multiple. Un EJB client supporté par le conteneur peut être un client HTTP, un conteneur Web ou un autre EJB.
- **maintient de la transparence de la localisation** : le conteneur maintient une transparence durant les appels, ainsi un client distant ne peut pas faire la distinction entre un appel local ou un appel distant.

En fournissant ces services, le conteneur aide le développeur à se concentrer seulement sur le développement de la logique business.

9) Qu'est ce qu'une entité ? Comment peut-on spécifier une clé primaire(éventuellement composite) au sein d'une entité ?

Les EJB entités permettent de représenter et de gérer des données enregistrées dans une base de données. Ils implémentent l'interface EntityBean.
L'avantage d'utiliser un tel type d'EJB plutôt que d'utiliser JDBC ou de développer sa propre solution pour mapper les données est que certains services sont pris en charge par le conteneur.
Les beans entité assurent la persistance des données en représentant tout au partie d'une table ou d'une vue.
Ainsi, un bean entité est une simple classe considérée comme POJO (Plain Old Java Object). Cela signifie qu'il faut que la classe comporte un constructeur par défaut. De plus, pour chaque attribut de la classe, un getter et un setter sont positionnés.
Un Bean entité aura comme annotation @Entity.
La clef primaire est définie par une annotation @Id.

Exemple :

```
@Entity
public class Customer // table = CUSTOMER
{
   private Long id;
   private String name;
   private Address address;

   @Id(generate=AUTO)// clé primaire
   public Long getID()
   {
      return id;
   }

   private void setID(Long id)
   {
      this.id = id;
   }

   ...
}
```

10) Comment une relation d'héritage peut-elle se traduire dans le modèle relationnel. Comment indique-t-on la manière de traduire cette relation d'héritage dans l'API de persistence ?

Avec la JPA (Java Persistance API), il est possible de choisir comment la classe doit persister lors de l'héritage. Pour chaque héritage, il est possible de choisir :

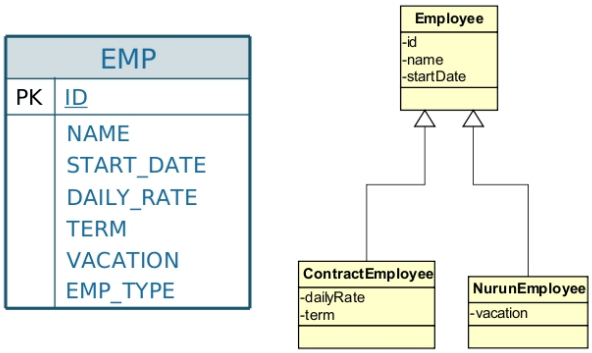
- une seule classe représente l'ensemble de l'héritage si celui-ci n'est pas trop complexe
- une classe par table concrète (une pour la table mère et une ou plusieurs pour les tables filles)
- une classe par table fille (et on reprend les informations de la table mère dans les tables filles)

On choisit le type d'héritage par l'annotation @Inheritance(...)

- strategy=InheritanceType.ONLY_ONE_TABLE (ou SINGLE_TABLE)
- strategy=InheritanceType.TABLE_PER_CLASS
- strategy=InheritanceType.JOINED

Une table pour une hiérarchie de classe :

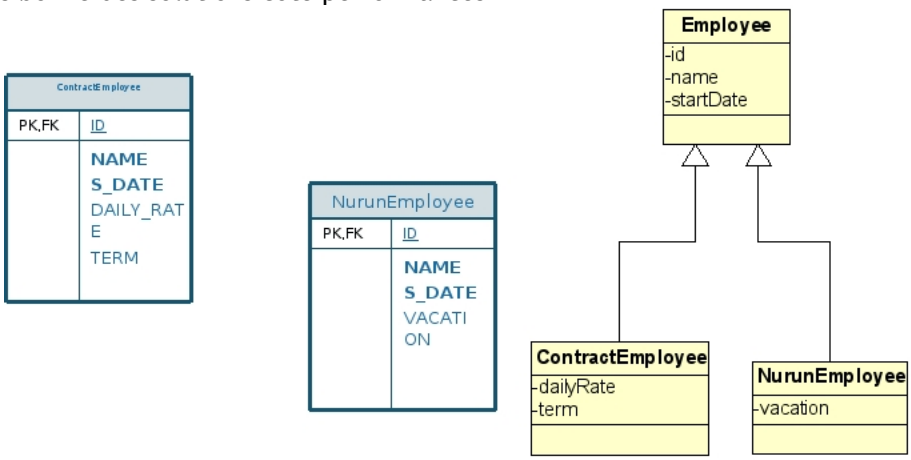
- toutes les propriétés de toutes les classes parentes et classes filles sont mappée dans la même table
- les instances sont différenciées par une colonne spéciale discriminante




```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn(
    name="EMP_TYPE",
    discriminatorType=DiscriminatorType.STRING
)

@DiscriminatorValue("Employee")
public class Employee { ... }

@Entity
@DiscriminatorValue("ContractEmployee")
public class ContractEmployee extends Employee { ... }
```

Une table par classe fille :

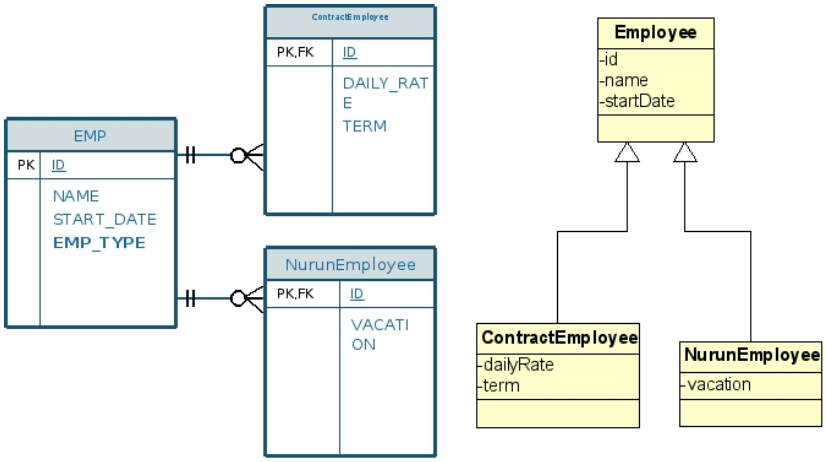
- une table pour chaque entité, et seulement des données de cette entité sont dans la table
- la moins bonne des solutions côté performances




```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class ContractEmployee implements Serializable { ...}
```

Une table par classe concrète :

- une table pour chaque entité, mais les propriétés communes sont regroupées
- bon compromis

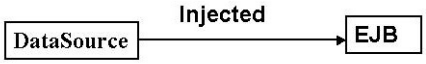



```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Employee implements Serializable { ... }

@Entity
public class ContractEmployee extends Employee { ... }

@Entity
@PrimaryKeyJoinColumn(name="ID") //id de l'employe
public class NurunEmployee extends Employee { ... }
```

11) Qu'est-ce que l'EntityManager ? Que permet-il de faire ? Comment peut-il être initialisé ?

L'EntityManager est une interface qui définit des méthodes pour interagir (gérer l'accès) avec le Persistence Context. Cette interface est utilisée pour créer et supprimer des instances d'entités persistantes, pour trouver des entités par leur clé primaire, et pour créer des requêtes sur ces entités. Le jeu d'entité qui peut être gérer par une instance d'un EntityManager donné est défini par une unité de persistance (Persistence Unit).

Container-Managed Entity Managers
Avec un Container-Managed Entity Managers, le contexte de persistance de l'instance d'un EntityManager est automatiquement propagée par le conteneur à tous les composants de l'application qui utilise l'instance de l'EntityManager grâce à une transaction JTA.
Un Container-Managed Entity Managers est obtenu dans une application à travers une injection de dépendance ou à travers un JNDI lookup.



Pour obtenir une instance d'un EntityManager, il faut « injecter » l'entity manager dans l'application :

```
@PersistenceContext
EntityManager em;
```

Le persistence context maintient l'unicité d'une instance en rapport avec l'identité de cette l'entité dans un contexte transactionnel.

Application-Managed Entity Managers
Avec Application-Managed Entity Managers, le contexte de persistance (persistence context) n'est pas propagé aux composants de l'application, et le cycle de vie de l'EntityManager est géré par l'application. Dans ce cas, chaque EntityManager crée un nouveau contexte de persistance isolé. L'EtityManager, et ses contextes de persistance associés, est créé et détruit explicitement par l'application.
Pour obtenir une instance d'un EntityManager, il faut utiliser le createEntityManger de l'EntityMangerFactory :

```
@PersistenceUnit
EntityManagerFactory emf;
EntityManager em = emf.createEntityManager();
```

Partie 2 : Web Services

1) De quoi est constitué un fichier WSDL ? Expliquer le rôle de chaque entrée. A quoi ce fichier sert-il ?

Le standard WSDL (Web Service Description Language) est un langage reposant sur la notation XML permettant de décrire les services web. WSDL permet ainsi de décrire l'emplacement du service web ainsi que les opérations (méthodes, paramètres et valeurs de retour) que le service propose.

Structure d'un document WSDL
Un document WSDL est un fichier au format XML utilisant les éléments suivants :

- **portType**, définissant le service web, en particulier les opérations qu'il réalise et le type de messages échangés.
- **message**, comprenant une ou plusieurs parties représentant les paramètres d'entrée.
- **types**, définissant les types de données utilisés par le service web.
- **binding**, précisant le protocole utilisé et le format de message.

Ce fichier est donc mis à la disposition de tout un chacun sur le serveur d'application et sera parsé par la première application qui aura besoin de se servir du module disponible sur le serveur. Une fois le fichier WSDL parcourut avec le bon outil, un proxy est créé afin de pouvoir communiquer avec le module se trouvant sur le serveur. Ce fichier permet également à une application cliente d'utiliser une représentation des méthodes du web service au travers du proxy. De cette manière, n'importe qui peut utiliser le web service distant en ayant l'impression qu'il tourne sur la même machine alors qu'en fait il s'agit d'une communication entre notre machine et le serveur.

Exemple :

CalculatorServiceServlet.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CalculatorServiceServlet" ...>
 <types/>
 <message name="CalculatorService_addition">
   <part name="int_1" type="xsd:int"/>
   <part name="int_2" type="xsd:int"/>
 </message>
 <message name="CalculatorService_additionResponse">
   <part name="result" type="xsd:int"/>
 </message>
 <portType name="CalculatorService">
   <operation name="addition" parameterOrder="int_1 int_2">
     <input message="tns:CalculatorService_addition"/>
     <output message="tns:CalculatorService_additionResponse"/>
   </operation>
 </portType>
 <binding name="CalculatorServiceBinding" type="tns:CalculatorService">
   ...
 </binding>
  <service name="CalculatorServiceServlet">
    <port name="CalculatorServicePort" binding="tns:CalculatorServiceBinding"/>
  </service>
</definitions>
```

2) Expliquez les étapes de développement que l'on peut être amené à réaliser pour coder un web service et son client.

WebService

1. Il faut écrire le web service proprement dit. Pour ce faire, il suffit d'écrire une classe précédée de l'annotation @webservice.
2. une fois compilé et transformé en fichier class, il faut utiliser un outil qui va créer ce qu'on appelle les artifacts. Ces artifacts sont en réalité d'autres fichiers sources qui permettent de rendre l'utilisation du WebService opérationnelle.
3. il faut empaqueter et déployer le WebService sur un serveur (ou utiliser une API qui permet de le faire). Pour ce faire, on va créer une archive WAR. Cette archive doit respecter une architecture bien particulière : dedans, il doit y avoir un répertoire WEB-INF dans lequel se trouvent 2 fichiers XML. Le premier généralement appelé web.xml sert à configurer la manière dont le WebService sera accessible depuis l'extérieur. Le second fichier sert à spécifier le nom et l'emplacement du WebService sur le serveur (sun-jaxws.xml). C'est ce qu'on appelle le « endpoint ».

Client

1. Pour commencer, il faut récupérer le fichier WSDL.
2. Une fois ce fichier récupéré, il faut générer un ensemble d'objets qui serviront à la communication entre le client et le WebService. Cette génération se fait au moyen de ce fichier WSDL. Une fois ce fichier parsé avec l'outil approprié, nous obtenons une série de classes qui correspondent à celles utilisées par le WebService. On a donc générer les artifacts.
3. On peut maintenant écrire le code du client en utilisant les objets générés grâce au fichier WSDL et ainsi utiliser notre WebService comme s'il s'agissait d'un objet local alors qu'il s'agit d'un objet distant, disponible sur un serveur.

3) Décrire les éléments d'une architecture RESTful. Qu'est ce que JAX-RS ? WADL ?

REST (Representational State Transfert) est un moyen d'accéder aux documents et aux ressources distants selon une architecture logicielle simple, représentée par une API.
Cette architecture se définit ainsi:

- Les états et fonctions d'une application distante sont considérées comme des ressources.
- Chaque ressource est accessible uniquement selon un format d'adresse standard. Ce sont des liens hypertextes (ou hypermedia). Sous HTTP se sera une URI.
- Les ressources ont une interface standard dans laquelle les opérations et types de données sont précisément définis. XML ou JSON par exemple, ou HTML.
- L'échange de données se fait selon un protocole qui a les propriétés suivantes:
  - Client-serveur.
  - Sans états (représentés par des variables).
  - Permet la mise en cache.
  - En couches logicielles multiples.
- Indépendance de l'interface aux services ajoutés tels que proxies, firewalls et autres.

JAX-RS est le nouveau JSR (Java Specification Requests) permettant d'implémenter une API RESTful en Java au moyen d'annotations. Il est très simple de déclarer quels URLs invoqueront quelles méthodes, quels paramètres (path, query...) seront acceptés et comment sérialiser les données en plusieurs formats (XML, JSON...).

WADL (Web Application Description Language,) permet de décrire les interfaces d'un service web RESTful :

- Définition des URIs
- Définition des paramètres
- Définition des ressources
- ...

Le WADL permet de décrire un Web Service de style RESTful dans un fichier XML.
Ils sont similaires aux fichiers WSDL qui sont destinés à décrire les Web Services de type SOAP.
Les fichiers WADL sont en général générés automatiquement par les outils qui construisent les Web Services.

4) Expliquez et illustrez à l'aide de lignes de code l'utilisation des SoapHeader pour pour une authentification. Quelles incidences cela a -t-il sur le fichier WSDL ?

SOAP Version 1.2 est un protocole pour l'échange d'information structurée dans un environnement distribué. Les messages SOAP sont spécifiés comme des XML Information Sets [XML InfoSet].SOAP ne dépend pas d'un langage de programmation
Nous allons illustrer l'utilisation des SOAP header à l'aide d'un WebService C#. Pour ce faire, il faut tout d'abord créer une classe (ici AuthHeader) qui hérite de SoapHeader et qui contiendra les informations nécessaire pour réaliser cette authentification.

```
namespace SystDistExam
{
    public class AuthHeader : SoapHeader
    {
        private string Login;
        public string getLogin
        {
            return Login;
        }
        public void setLogin(string Login)
        {
            this.Login = Login;
        }
    }
}
```

Dans notre WebService, nous devons préciser à la méthode visée que l'on exige d'avoir une authentification de la part du client pour pouvoir l'exécuter.

```
namespace SystDistExam
{
 ...
    [System.Web.Script.Services.ScriptService]
    public class Exam : System.Web.Services.WebService
    {
        public AuthHeader loginheader;
        ...
        [SoapHeader("loginheader")]
        public MensualitePret[]
        CalculEcheance(double Capital, int Duree, double TauxAnnuel)
        {
            if (loginheader.Login == "toto")
            {
                //code de la fonction
            }
        }
}
```

Incidence sur le code WSDL

```
</s:complexType>
<s:element name="AuthHeader" type="tns:AuthHeader"/>
<s:complexType name="AuthHeader">
  <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Login" type="s:string"/>
  </s:sequence>
  <s:anyAttribute/>
</s:complexType>
…
<wsdl:message name="CalculEcheanceAuthHeader">
<wsdl:part name="AuthHeader" element="tns:AuthHeader"/>
</wsdl:message>
…
<wsdl:binding name="Labo3Soap" type="tns:Labo3Soap">

<wsdl:input>
<soap:body use="literal"/>
<soap:header message="tns:CalculEcheanceAuthHeader" part="AuthHeader"
use="literal"/>
</wsdl:input>
```

5) Citez et expliquez deux méthodes vues au cours pour rendre disponible en Java un Web Service ?

Self-hosting

Cette méthode consiste à publier le web service au travers d'une application java qui se chargera de créer et publier les endpoints nécessaire. Ce sera donc une classe de Java qui gérera tout le mécanisme pour nous afin de nous faciliter la tâche.

```
package WebApplication.server;

import javax.xml.ws.Endpoint;

public class AddWebService
{
  public static void main (String[] args) throws Exception
  {
     Endpoint endpoint = Endpoint.publish ("http://localhost:8080/jaxws-
     WebApplication/getipinfo",new GetIPInfoImpl());

     // Stops the endpoint if it receives request http://localhost:9090/stop
     new EndpointStopper(9090, endpoint);
  }
}
```

Glassfish

Héberger un web service sous glasshfish est des plus simples. Il suffit de publier via l'interface de glasshfish le war de l'application. Ceci aura pour effet de mettre en place le contexte d'exécution et d'exposé le code WSDL de l'application.

Partie 3 : WCF

1) Citez et expliquez deux méthodes pour rendre disponible un service WCF.

La publication sur un serveur IIS

Internet Information Service est le nom du serveur web de Microsoft. Ce mode d'hébergement est très utilisé, car il est très simple à mettre en œuvre (un fichier d'une ligne suffit à héberger le service) et contrairement à l'auto-hébergement, le service démarrera automatiquement avec le serveur Web. Il ne nécessite donc pas d'attention particulière.
Cependant, de par sa nature même de serveur web, IIS ne supporte que les protocoles http et https. Bien que la majorité des services WCF en production soient hébergés à l'aide de ces protocoles, cela peut être une limite dans certains cas.
Enfin, on notera que l'hébergement d'un service WCF sous IIS permet de profiter facilement de l'authentification IIS/ASP.NET et de la gestion des profils

Le self-hosting

Le self-Hosting désigne le fait que le service WCF soit hébergé à l'intérieur d'une application Windows. Le nom « self-hosting » vient du fait que dans ce cas de figure c'est à l'application d'implémenter elle-même tous les mécanismes de gestion des canaux de communication (ouverture, recyclage, fermeture). Heureusement, le Framework .Net facilite la tâche du développeur à travers la classe ServiceHost.

```
ServiceHost host = new ServiceHost(typeof(EvalService));
           host.Open();
```

L'auto hébergement est en réalité la façon la plus simple de mettre à disposition un service WCF. En effet, contrairement aux autres solutions, elle ne nécessite comme pré requis que d'avoir le Framework .Net 3.0 ou supérieur d'installé. En contrepartie c'est la solution qui propose le moins de fonctionnalités annexes. C'est au développeur de tout gérer lui-même.
De plus, l'auto hébergement n'est probablement pas une bonne solution dans le cas d'un service en production. En effet, exception faite de l'hébergement dans un service Windows, il sera toujours nécessaire de lancer l'application à la main après avoir ouvert une session sur la machine, chose que l'on évite généralement de faire en production, car en cas de redémarrage du serveur, le service WCF sera indisponible tant qu'un opérateur ne l'aura pas relancé.

2) Qu'est-ce qu'un endpoint WCF configuré dans un serviceModel ? Citez et expliquez deux types de binding prédéfinis dans WCF. Où sont-ils spécifiés pour un service donné ?

Chaque service doit avoir une adresse qui définit où est situé le service, un contrat qui définit ce que fait le service et un « binding » qui définit comment communiquer avec le service. Dans un WCF, la relation entre une adresse, un contrat et un « binding » est appelé endpoint.

Un service WCF peut exposer un ou plusieurs points de communication (ou « endpoint »), chacun d'eux étant constitué d'une adresse (A), d'un binding (B) et d'un contrat (C):

- L' « **adresse** » spécifie la localisation du service. Elle est définie par une URI renseignée dans le code ou déclarée dans le fichier de configuration. Un même service WCF peut exposer plusieurs points de communication et donc être accessible depuis plusieurs adresses.
- Le « **binding** » permet de décrire le protocole utilisé pour se connecter à un point de communication d'un service WCF en spécifiant chaque élément de la pile de communication. C'est cette partie qui est la plus important de l'endpoint. Il se compose de deux points fondamentaux :
  - Le **protocole de transport** : il est nécessaire d'utiliser des protocoles pour transporter les données afin que les différentes machines qui dialoguent entre elles se comprennent.
  - Le **format des données** : Le format des données sert, comme son nom l'indique, à définir comment le client et le serveur vont dialoguer, comment les requêtes du client doivent être formées et comment les réponses du serveur doivent être interprétées.

  En quelque sorte, le binding défini la langue dans laquelle le serveur et le client vont communiquer.
- Le « **contrat** » permet au service de décrire explicitement la liste des opérations qu'il expose. La définition du contrat consiste à déclarer une interface et à définir un certain nombre de méthodes en les « décorant » avec des attributs prédéfinis (certains de ces attributs permettent également de spécifier un contrat de service incluant des caractéristiques telles que l'échange bidirectionnel, le support d'une session pour conserver l'état entre l'appel de plusieurs opérations, l'attente ou non d'un message d'acquittement suite à l'exécution d'une opération ne retournant pas de résultat,....)

Bindings :
Par défaut, WCF propose 9 « binding » standards :

- « **BasicHttpBinding** » : Représente une liaison qu'un service WCF peut utiliser pour configurer et exposer des points de terminaison capables de communiquer avec des clients et services Web basés sur ASMX. La sécurité est désactivée par défaut. Ses options de sécurité sont très limitées.
- « **NetTcpBinding** » : un « binding » sécurisé et optimisé adapté à la communication inter machines entre applications WCF
- « **WSHttpBinding** » : représente une liaison interopérable qui prend en charge les transactions distribuées et les sessions fiables et sécurisées. Il est semblable à BasicHttpBinding mais fournit plus de fonctionnalités de service Web. Il utilise le transport HTTP et assure la sécurité des messages, comme BasicHttpBinding, mais il fournit également des transactions, une messagerie fiable et WS-Addressing, qu'il soit actif par défaut ou disponible par l'intermédiaire d'un paramètre de contrôle unique.

La définition des « bindings » peut être spécifiée par code ou par configuration. Par exemple, si l'on considère le « binding » « BasicHttpBinding », le constructeur de la classe « BasicHttpBinding » associée permet de créer directement un « binding » sécurisé en prenant en paramètre l'un des modes de sécurité décrit dans l'énumération dédiée à ce « binding » : « BasicHttpSecurityMode ». Il est également possible de sécuriser ce « binding » après sa construction, comme l'illustre l'exemple suivant :

```
BasicHttpBinding binding = new BasicHttpBinding();
binding.Security.Mode = BasicHttpSecurityMode.Transport;
binding.Security.Transport.ClientCredentialType =
HttpClientCredentialType.None;
host.AddServiceEndpoint(typeof(EspaceServices.LeService),binding,"https://serve
ur:666/LeService");
```

Il est également possible de définir la sécurité du «binding» par configuration. L'exemple suivant présente la configuration appropriée pour un «basicHttpBinding» et un mode de sécurité transport.
La définition du «endpoint» spécifie le «basicHttpBinding» et référence une configuration de «binding» nommée «secured».

```
<services>
   <service name="EspaceServices.LeService">
      <endpoint
         address=https://serveur:666/LeService
         binding="basicHttpBinding" bindingConfiguration="secured"
         contract="EspaceServices.ILeService"
      />
   </service>
</services>
<bindings>
   <basicHttpBinding>
      <binding name="secured">
          <security mode="Transport">
              <transport clientCredentialType="None"/>
          </security>
      </binding>
   </basicHttpBinding>
</bindings>
```

3) Quels sont les différents types de « contracts » qu'un service WCF doit spécifier ? Quels sont leur rôles ?Dans quelle dll sont -ils définis ?

Un service WCF doit connaitre au minimum un contrat qui est le contrat de service. Néanmoins, si un objet métier est nécessaire au bon fonctionnement du service, il devra spécifier également un contrat de donnée qui porte sur la classe en question.

Un **contrat de données** : il définit le format des données d' un objet métier (classe) / structure / énumération qui sera échangé entre le service et le client. Il est définit par l'attribut [DataContract] et ses membres marqués de l'attribut [DataMember] (Si un membre n'est pas balisé [DataMember] il ne sera pas sérialisé dans le flux XML qui sera échangé entre le service et le client).

Un **contrat de service** : interface qui définit les méthodes (=opérations) que le service WCF va exposer aux clients. Il est définit par l'attribut [ServiceContract] et ses membres marqués de l'attribut [OperationContract].

Ils sont définis dans « System.ServiceModel.dll ».