Programmation orientée objet en langage JAVA

Java Naming and Directory Interface et Lightweight Directory Access Protocol

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Java Naming and Directory Interface

Introduction à JNDI (1/3)

- JNDI est l'acronyme de Java Naming and Directory Interface.
- Cette API fournit une interface unique pour utiliser différents services de nommages ou d'annuaires et définit une API standard pour permettre l'accès à ces services.
- Il existe plusieurs types de service de nommage parmi lesquels :
  - DNS (Domain Name System) : service de nommage utilisé sur internet pour permettre la correspondance entre un nom de domaine et une adresse IP.
  - LDAP(Lightweight Directory Access Protocol) : annuaire.
  - NIS (Network Information System) : service de nommage réseau développé par Sun Microsystems.
  - COS Naming (Common Object Services) : service de nommage utilisé par Corba pour stocker et obtenir des références sur des objets Corba.
  - etc.

Introduction à JNDI (2/3)

- Un service de nommage permet d'associer un nom unique à un objet et faciliter ainsi l'obtention de cet objet.
- Un annuaire est un service de nommage qui possède en plus une représentation hiérarchique des objets qu'il contient et un mécanisme de recherche.
- JNDI propose donc une abstraction pour permettre l'accès à ces différents services de manière standard.
- Ceci est possible grâce à l'implémentation de pilotes qui mettent en œuvre la partie SPI de l'API JNDI.
- Cette implémentation se charge d'assurer le dialogue entre l'API et le service utilisé.

Introduction à JNDI (3/3)

- JNDI possède un rôle particulier dans les architectures applicatives développées en Java car elle est utilisée dans les spécifications de plusieurs API majeures : JDBC, EJB, JMS, ...
- De plus, la centralisation de données dans une source unique pour une ou plusieurs applications facilite l'administration de ces données et leur accès.
- Pour plus d'informations sur JNDI : `http://java.sun.com/products/jndi`.
- Sun propose un excellent tutorial sur JNDI à l'url : `http://java.sun.com/products/jndi/tutorial/`.
- Pour utiliser JNDI, il faut un service de nommage correctement installé et configuré, et un pilote dédié à ce service.

Présentation de JNDI (1/3)

- JNDI est composée de deux parties :
  - Une API utilisée pour le développement des applications.
  - Une SPI utilisée par les fournisseurs d'une implémentation d'un pilote.
- Un pilote est un ensemble de classes qui implémentent les interfaces de JNDI pour permettre les interactions avec un service particulier.
- Ce mode de fonctionnement est identique à celui proposé par l'API JDBC.
- Il est donc nécessaire de disposer d'un pilote pour assurer le dialogue entre l'application via l'API et le service de nommage ou l'annuaire.
- La partie API est incluse dans le JDK et Sun propose une implémentation des pilotes pour LDAP, DNS et Corba.

Présentation de JNDI (2/3)

- Pour utiliser d'autres services ou d'autres implémentations, il faut utiliser les implémentations des pilotes fournis par des fournisseurs tiers.
- Pour définir une connexion, JNDI à besoin d'au moins deux éléments :
  - La fabrique du contexte racine : c'est cet objet qui assure le dialogue avec le service utilisé en utilisant le protocole adéquat.
  - L'url du service à utiliser.
- JNDI n'est pas utilisable uniquement pour des applications J2EE.
- Une application standalone peut par exemple réaliser une authentification à partir d'un annuaire via le protocole LDAP.
- Ainsi JNDI est inclus dans J2SE depuis la version 1.3 du J2SE.

Présentation de JNDI (3/3)

- Pour les versions antérieures (J2SE 1.1 et 1.2), il est nécessaire de télécharger JNDI en tant qu'extension standard et de l'installer.
- Il est aussi possible d'utiliser d'autres pilotes fournis séparément par Sun ou par d'autres fournisseurs.
- Sun propose une liste des pilotes existant à l'url : `http://java.sun.com/products/jndi/serviceproviders.html`
- Sun propose aussi le "JNDI/LDAP Booster Pack" qui propose des pilotes pour des serveurs LDAP et un pilote permettant la mise en œuvre de DSML (Directory Services Markup Language) dont le but est d'accéder à un annuaire avec XML.

Les services de nommage (1/2)

- Il existe de nombreux services de nommage : les plus connus sont sûrement les systèmes de fichiers (File system), les DNS, les annuaires LDAP, ...
- Un service de nommage permet d'associer un nom à un objet ou à une référence sur un objet.
- L'objet associé dépend du service : un fichier dans un système de fichiers, une adresse I.P. dans un DNS, ...

Les services de nommage (2/2)

- Le nom associé à un objet respecte une convention de nommage particulière à chaque type de service :
  - Avec un système de fichiers de type Unix, le nom est composé d'éléments séparés par un caractère "/".
  - Avec un système de fichiers de type Windows, le nom est composé d'éléments séparés par un caractère "\".
  - Avec un service de type DNS, le nom est composé d'éléments séparés par un caractère "." (exemple : www.toto.fr).
  - Avec un service de type LDAP, le nom désigné par le terme Distinguished Name est composé d'élément séparé par un caractère ",".
  - Un élément est de la forme clé=valeur.
- Pour permettre une abstraction des différents formats de noms utilisés par les différents services, JNDI utilise la classe `Name`.

Les annuaires (1/2)

- Un annuaire est un outil qui permet de stocker et de consulter des informations selon un protocole particulier.
- Un annuaire est plus particulièrement dédié à la recherche et la lecture d'informations : il est optimisé pour ce type d'activité mais il doit aussi être capable d'ajouter et de modifier des informations.
- Les annuaires sont des extensions des services de nommage en ajoutant en plus la possibilité d'associer d'éventuels attributs à chaque objet.

Caractéristiques	**Annuaire**	**Bases de données**
Accès aux données	Lecture privilégiée	Lecture et modification
Représentation des données	Hiérarchique	Ensembliste

Les annuaires (2/2)

- Les annuaires les plus connus dans le monde réel sont les pages jaunes et les pages blanches du principal opérateur téléphonique.
- Le but de ces deux annuaires est identique (obtenir un numéro de téléphone) mais la structure des données est différente :
  - Pages blanches : regroupement par département, ville, nom/prénom.
  - Pages jaunes : regroupement par activités, ville, nom.
- Les systèmes de fichiers sont aussi des annuaires : ils associent un nom à un fichier mais stockent aussi des attributs liés à ces fichiers (droits d'accès, dates de création et de modification, ...)

Le contexte

- Un service de nommage permet d'associer un nom à un objet. Cette association est nommée binding. Un ensemble d'associations nom/objet est nommé un contexte.
- Ce contexte est utilisé lors de l'accès à un élément contenu dans le service.
- Il existe deux types de contexte :
  - Contexte racine,
  - Sous-contexte.
- Un sous-contexte est un contexte relatif à un contexte racine.
- Par exemple, "c :\" est un contexte racine dans un système de fichiers de type Windows.
- Le répertoire windows est un sous-contexte du contexte racine (« c :\windows ») qui est dans ce cas nommé sous répertoire.
- Dans le DNS, `com` est un contexte racine et `test` est un sous-contexte (`test.com`)

La mise en œuvre de l'API JNDI

- L'API JNDI est contenue dans cinq packages :
  - **javax.naming :** Classes et interfaces pour utiliser un service nommage.
  - **javax.naming.directory :** Étend les fonctionnalités du package javax.naming pour l'utilisation des services de type annuaire.
  - **javax.naming.event :** Classes et interfaces pour la gestion des événements lors d'un accès à un service.
  - **javax.naming.ldap :** Étend les fonctionnalités du package javax.naming.directory pour l'utilisation de la version 3 de LDAP.
  - **javax.naming.spi :** Classes et interfaces dédiées aux Service Provider pour le développement de pilotes.

L'interface `Name`

- Cette interface encapsule un nom en permettant de faire abstraction des conventions de nommage utilisées par le service.
- Deux classes implémentent cette interface :
  - CompositeName : chaque élément qui compose le CompositeName est séparé par un caractère /.
  - CompoundName : chaque élément issu de la hiérarchie compose le nom selon certaines règles dépendantes de l'implémentation.

L'interface `javax.Naming.Context`

- Elle représente un ensemble de correspondances nom/objet d'un service de nommage.
- Elle propose des méthodes pour interroger et mettre à jour ces correspondances.
  - **void bind(String, Object) :** Ajouter une nouvelle correspondance entre le nom et l'objet passé en paramètre.
  - **void rebind(String, Object) :** Refaire la correspondance.
  - **Object lookup(String) :** Renvoie un objet à partir de son nom.
  - **void unbind(String) :** Supprimer la correspondance désignée par le nom fourni en paramètre.
  - **void rename(String, String) :** Modifier le nom d'une correspondance.
  - **NamingEnumeration listBindings(String) :** Renvoie les objets associés à la correspondance dont le nom est fourni en paramètre.

La classe `InitialContext` (1/3)

- Elle implémente l'interface `Context` et encapsule le contexte racine : c'est le nœud qui sert de point d'entrée lors de la connexion avec le service.
- Toutes les opérations réalisées avec JNDI sont relatives à ce contexte racine.
- Cette interface encapsule le point d'entrée dans le service de nommage.
- Pour obtenir une instance de la classe InitialContext et ainsi réaliser la connexion au service, plusieurs paramètres sont nécessaires :
  - java.naming.factory.initial permet de préciser le nom de la fabrique proposée par le fournisseur. Cette fabrique est en charge de l'instanciation d'un objet de type InitialContext
  - java.naming.provider.url : URL du context racine.

La classe `InitialContext` (2/3)

- Plusieurs fabriques sont fournies en standard depuis le J2SE 1.4 :

CORBA	com.sun.jndi.cosnaming.CNCtxFactory
DNS	com.sun.jndi.dns.DnsContextFactory
LDAP	com.sun.jndi.ldap.LdapCtxFactory
RMI	com.sun.jndi.rmi.registry.RegistryContextFactory

- Les deux paramètres (`java.naming.factory.initial` et `java.naming.provider.url`) sont obligatoires mais d'autres peuvent être nécessaire notamment ceux concernant la sécurité pour l'accès au service.

La classe `InitialContext` (3/3)

- L'interface Context définit des constantes pour le nom de ces paramètres.
- Il y a plusieurs moyens pour définir ces paramètres :
  - les définir sous la forme de variables d'environnement passées à la JVM en utilisant l'option `-D`.
  - les définir sous la forme d'une collection de type Hashtable passée en paramètre au constructeur de la classe `InitialContext`.
  - les définir dans un fichier nommé jndi.properties accessible dans le classpath.
- Exemple :

```
Hashtable hashtableEnvironment = new Hashtable();
hashtableEnvironment.put(Context.INITIAL_CONTEXT_FACTORY,
                         "com.sun.jndi.fscontext.RefFSContextFactory");
hashtableEnvironment.put(Context.PROVIDER_URL, "file:c:/");
Context context = new InitialContext(hashtableEnvironment);
```

L'utilisation d'un service de nommage

- Pour pouvoir utiliser un service de nommage, il faut tout d'abord obtenir un contexte racine qui va encapsuler la connexion au service.
- À partir de ce contexte, il est possible de réaliser plusieurs opérations :
  - bind : associer un objet avec un nom.
  - rebind : modifier une association.
  - unbind : supprimer une association.
  - lookup : obtenir un objet à partir de son nom.
  - list : obtenir une liste des associations.
- Toutes les opérations possèdent deux versions surchargées attendant respectivement :
  - Un objet de type Name : cet objet encapsule une séquence ordonnée d'un ou plusieurs éléments (l'intérêt de cette classe est de permettre la manipulation individuel de chaque élément).
  - Une chaine de caractères : elle contient la séquence.

L'obtention d'un objet

- Pour obtenir un objet du service de nommage, if faut utiliser la méthode lookup() du contexte.

```
import javax.naming.*;

public String getValeur() throws NamingException {
    Context context = new InitialContext();
    return (String) context.lookup("/config/monApplication");
}
```

- Ceci peut permettre de facilement stocker des options de configuration d'une application, plutôt que de les stocker dans un fichier de configuration.
- Ceci est d'autant plus intéressant si le service qui stocke ces données est accessible via le réseau car cela permet de centraliser ces options de configuration.
- Il peut permettre aussi de stocker des données "sensibles" comme des noms d'utilisateur et des mots de passe pour accéder à une ressource.

Le stockage d'un objet

- Généralement les objets à stocker doivent être d'un type particulier, dépendant du pilote utilisé : il est fréquent que de tels objets doivent implémenter une interface (`java.io.Serializable`, `java.rmi.Remote`, etc.)
- La méthode bind() permet d'associer un objet à un nom.
- Exemple :

```
import  javax.naming.*;

public void createName() throws NamingException {
    Context context = new InitialContext();
    context.bind("/config/monApplication", "valeur");
}
```

Les annuaires LDAP

Présentation de LDAP (1/6)

- LDAP, acronyme de Lightweight Directory Access Protocol, est un protocole de communication vers un annuaire en utilisant TCP/IP.
- Le but principal est de retrouver des données insérées dans l'annuaire.
- Ce protocole est optimisé pour la lecture et la recherche d'informations.
- LDAP est un protocole largement supporté par l'industrie informatique.
- Il existe de nombreuses implémentations libres et commerciales : Microsoft Active Directory, OpenLDAP, Netscape Directory Server, Sun NIS, Novell NDS, ..
- Ce protocole ne précise pas comment ces données sont stockées sur le serveur.

Présentation de LDAP (2/6)

- Un serveur de type LDAP peut stocker n'importe quel type de données : ce sont souvent des ressources (personnes, matériels réseaux, ...).
- La version actuelle de LDAP est la v3 définie par les RFC 2252 et RFR 2256 de l'IETF.
- Dans un annuaire LDAP, les nœuds sont organisés sous une forme arborescente hiérarchique nommée le DIT (Direct Information Tree).
- Chaque nœud de cette arborescence représente une entrée dans l'annuaire.
- Chaque entrée contient un objet qui possède un ou plusieurs attributs dont les valeurs permettent d'obtenir des informations sur l'objet.
- Un objet appartient à une classe au sens LDAP.

Présentation de LDAP (3/6)

- La première et unique entrée dans l'arborescence est nommée racine.
- Chaque objet possède un Relatif Distinguish Name (RDN) qui correspond à une paire clé/valeur d'un attribut obligatoire.
- Un objet est identifié de façon unique grâce à référence unique dans le DIT : son Distinguish Name (DN) qui est composé de l'ensemble des RDN de chaque objet père dans l'arborescence lu de droite à gauche et son RDN (ceci correspond donc au DN de l'entrée père et de son RDN).
- Cette référence représente donc le chemin d'accès depuis la racine de l'arborescence. Le DN se lit de droite à gauche puisque la racine est à droite.
- La convention de nommage utilisée pour le DN, utilise la virgule comme séparateur et se lit de droite à gauche.

Présentation de LDAP (4/6)

- Exemple :
  `uid=jm,ou=utilisateur,o=test.com`
- Le premier élément du DN, nommé Relative Distinguished Name (RDN), est composé d'une paire clé/valeur.
- Comme valeur de clé, LDAP utilise un mnémonique :

dn	Distinguished name	Nom unique dans l'arborescence
uid	Userid	Identifiant unique pour l'utilisateur
cn	Common name	Nom et prénom d'un utilisateur
givenname	First name	Prénom d'un utilisateur
sn	Surname	Nom de l'utilisateur
l	Location	Ville de l'utilisateur
o	Organization	Généralement la racine de l'annuaire
ou	Organizational unit	Généralement une branche de l'arbre
st	State	Etat du pays de l'utilisateur
c	Country	pays de l'utilisateur
Mail	Email	Email de l'utilisateur

Présentation de LDAP (5/6)

- Un élément qui compose une entrée dans l'annuaire est nommé objet.
  - Chaque objet peut contenir des attributs obligatoires ou facultatifs.
  - Un attribut correspond à une propriété d'un objet, par exemple un email ou un numéro de téléphone pour une personne.
  - Un attribut se présente sous la forme d'une paire clé/valeur(s).
- Les classes caractérisent les objets en définissant les attributs optionnels et obligatoires qui les composent.
- Il existe des attributs standards communément utilisés mais il est aussi possible d'en définir d'autre.
- L'ensemble des règles qui définissent l'arborescence et les attributs utilisables sont stockés dans un schéma.
  - Un schéma permet donc de définir les classes et les objets pouvant être stockées dans l'annuaire.
  - Un annuaire pour supporter plusieurs schémas.

Présentation de LDAP (6/6)

- Une fonctionnalité intéressante des annuaires est la possibilité de pouvoir stocker des objets Java directement dans l'annuaire et de pouvoir les retrouver en utilisant le protocole LDAP.
- Ces objets peuvent avoir des fonctionnalités diverses telle qu'une connexion à une source de données, à un objet contenant des options de paramétrage de l'application, etc ...
- Un serveur LDAP propose les fonctionnalités de base suivantes :
  - Connexion/déconnexion au serveur
  - Gestion de la sécurité lors d'accès aux objets
  - Ajout, modification, suppression d'objets
  - Gestion d'attributs sur les objets
  - Recherche d'objets

OpenLDAP (1/3) : version ancienne

- Vous pouvez facilement télécharger et installer OpenLDAP.
  `sudo apt-get install slpad ldap-utils`
- OpenLDAP propose en standard plusieurs schémas prédéfinis stockés dans le sous répertoire schema.
- Le fichier `slapd.conf` contient les principaux paramètres.
- Il est installé pré-paramétré lors de l'installation dans le répertoire d'installation d'OpenLDAP.
- Au début du fichier, il faut ajouter le schéma java pour utiliser openldap avec JNDI pour stocker des objets java.
- Exemple :

```
ucdata-path       ./ucdata
include           ./schema/core.schema
include           ./schema/java.schema
```

OpenLDAP (2/3) : version ancienne

- Il faut ensuite configurer la base de données, le suffixe qui est la racine du serveur et le compte de l'administrateur du serveur (root).
- Exemple :

```
#######################################################################
# BDB définition de la base de données
#######################################################################
database        bdb
suffix          "dc=mondomaine,dc=net"
rootdn          "cn=manager,dc=mondomain,dc=net"
rootpw          motdepasse
directory       ./data
index           objectClass     eq
```

OpenLDAP (3/3) : version ancienne

- Il faut remplacer la valeur des clés suffix et rootdn par les valeurs appropriés au contexte.
- Exemple :

```
suffix              "dc=test-ldap,dc=net"
rootdn              "cn=ldap-admin,dc=test-ldap,dc=net"
```

- Pour insérer le mot de passe dans le fichier slapd.conf, il faut le crypter grâce à la commande slappasswd
- Exemple :

```
slappasswd -s ldap-admin
{SSHA}ZUPUkq7mt21rEmrFgFc0cgk9izpwL7oY
```

- Il suffit alors de remplacer dans le fichier slapd.conf la ligne

```
rootpw secret
```

- par la ligne ci dessous qui contient le mot de passe crypté

```
rootpw {SSHA}ZUPUkq7mt21rEmrFgFc0cgk9izpwL7oY
```

- Il faut ensuite démarrer ldap.

OpenLDAP (1/2) : nouvelle version

- Il n'y a pas de fichier `slapd.conf` mais un fichier `ldap.conf` et un répertoire `slapd.d` qui contiennent les paramètres de configuration.
- Pour lancer la configuration, il faut taper la commande suivante :
  `sudo dpkg-reconfigure slapd`
- Puis il faut répondre aux questions de configuration :
  1. Passer la configuration d'OpenLDAP ? non.
  2. Nom de domaine ? example.com.
  3. Nom de votre société ? masociété.
  4. Quelle base de donnée ? hdb.
  5. Voulez-vous que la base de données soit effacée lorsque slapd est purgé ? oui.
  6. Supprimer les anciennes bases de données ? oui.
  7. Mot de passe administrateur ? VotreMotDePasse.
  8. Confirmer ce mot de passe ? VotreMotDePasse.
  9. Autoriser le protocol LDAPv2 ? non.

OpenLDAP (2/2) : nouvelle version

- Il faut ensuite éditer ou créer le fichier `/etc/ldap/ldap.conf` et ajouter les lignes suivantes :

```
ldap_version 3
URI ldap://localhost:389
SIZELIMIT 0
TIMELIMIT 0
DEREF never
BASE dc=example, dc=net
```

PhpLdapAdmin

- Il s'agit d'un outil développer en PHP pour gérer et administrer un serveur LDAP.
- Pour l'installer, il suffit d'installer le package : `sudo apt-get install phpldapadmin`.
- Pour se connecter, il vous faudra renseigner l'identifiant `cn=admin,dc=ldap,dc=example,dc=com` et le mot de passe renseigné précédemment.
- Le fichier de configuration s'appelle `config.php` et il est situé dans le répertoire `/etc/phpldapadmin`.

Le langage LDIF (1/3)

- Le format LDIF permet de réaliser des opérations d'import/export de données d'un annuaire.
- Le format général de ce format est le suivant :

```
[<id>]
dn: <distinguished name>
objectclass: <objectclass>
objectclass: <objectclass>
...
<attribut> : <valeur>
<attribut> : <valeur>
...
```

- Chaque entrée est séparée dans le fichier par une ligne vide.
- `<id>` est un entier positif facultatif qui représente un identifiant dans les données du serveur.
- Chaque élément définit dans le fichier est séparé par une ligne vide. Il commence par son DN

Le langage LDIF (2/3)

- Chaque attribut est définit sur sa propre ligne.
- La définition peut se poursuivre sur la ligne suivante si celle-ci commence par une espace ou une tabulation.
- Pour fournir plusieurs valeurs à un attribut, il suffit de répéter la clé de cet attribut sur une ligne pour chaque valeur.
- Si la valeur d'un attribut contient des caractères non imprimables (des données binaires comme une image par exemple) alors la clé de l'attribut est suivi de : : et la valeur est encodé en base 64.
- Le format LDIF permet également d'effectuer des modifications de données grâce à des opérations : add (ajouter une entrée), delete (supprimer une entrée), modrdn (modifier le rdn).

Le langage LDIF (3/3)

- Exemple : Le fichier `test.ldif`

```
dn: dc=test-ldap,dc=net
objectClass: dcObject
objectClass: organization
dc: test-ldap
o: Enreprise Test
description: Entreprise de tests

dn: cn=Durand,dc=test-ldap,dc=net
objectClass: organizationalRole
cn: Durand
description: Président directeur général
```

L'interface `DirContext` (1/2)

- L'interface DirContext est une classe fille de l'interface Context.
- Elle propose des fonctionnalités pour utiliser un service de nommage et propose en plus des fonctionnalités dédiées aux annuaires telles que la gestion des attributs et la recherche d'éléments.
  - **void bind (String, Object, Attributes)**
    Permet d'associer un objet avec des attributs à un nom.
  - **void rebind (String, Object, Attributes)**
    Permet de modifier l'association d'un objet.
  - **Attributes getAttributes(String)**
    Permet d'obtenir tous les attributs de l'objet associé au nom fourni en paramètre.
  - **Attributes getAttributes(String, String [])**
    Permet d'obtenir les attributs de l'objet associé au nom fourni en paramètre.
    Seuls les attributs fournis en paramètres sont retournés par la méthode.

L'interface `DirContext` (2/2)

- Pour pouvoir accéder à un annuaire, les étapes sont similaires à celles d'un accès à un service de nommage.
- Il faut obtenir une instance de type DirContext en instanciant un objet de type InitialDirContext().
- Cet objet a besoin de paramètres généralement fournis sous la forme d'une collection de type Hashtable.
- Ces paramètres sont les mêmes que pour un accès à un service de nommage.

La classe `InitialDirContext` (1/3)

- L'instanciation d'un objet de type InitialDirContext permet de se connecter à l'annuaire et de se positionner à un endroit précis de l'arborescence de l'annuaire nommé contexte initial.
- Toutes les opérations alors réalisées dans l'annuaire le seront relativement à ce contexte initial.
- Pour se connecter à un serveur LDAP, il faut obtenir un objet qui implémente l'interface DirContext : c'est généralement un objet de type InitialDirContext qui est obtenu en utilisant une collection de type Hashtable contenant les paramètres de connexion fourni à une fabrique dédiée.

La classe `InitialDirContext` (2/3)

- Afin de permettre de réaliser la connexion, il est nécessaire de fournir des paramètres pour configurer son environnement.
- Ces paramètres sont fournis au constructeur de la classe InitialDirContext sous la forme d'un objet de type Hashtable : ces paramètres concernent plusieurs types d'informations :
  - Le fournisseur de l'implémentation.
  - La localisation de l'annuaire.
  - La sécurité d'accès.
- Deux paramètres sont obligatoires :
  - `Context.INITIAL_CONTEXT_FACTORY` permet de préciser la classe fournie par le fournisseur
  - `Context.PROVIDER_URL` permet de préciser une url pour localiser l'annuaire. Le format de cette url dépend du fournisseur
- Exemple :

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,"ldap://localhost:389");
DirContext dircontext = new InitialDirContext(env);
```

La classe `InitialDirContext` (3/3)

- Si l'accès au serveur est sécurisé, il faut fournir des paramètres supplémentaires pour permettre cette authentification : le type de sécurité utilisée, le DN d'un utilisateur et son mot de passe :
  - `Context.SECURITY_AUTHENTICATION` Permet de préciser le type de sécurité utilisé.
    Les valeurs possibles sont : simple, SSL, SASL
  - `Context.SECURITY_PRINCIPAL` Permet de préciser le Distinguished Name de l'utilisateur
  - `Context.SECURITY_CREDENTIALS` Le mot de passe de l'utilisateur
- LDAP supporte trois modes de sécurité :
  - **Simple :** pas de cryptage du DN de l'utilisateur ni de son mot de passe
  - **SSL :** utilisation du cryptage SSL à travers le réseau si le serveur LDAP le supporte
  - **SASL :** utilisation des algorithmes MD5/Kerberos

Exemple d'utilisation LDAP (1/2)

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;

public class TestLDAP {

  public static void main(String[] args) {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://localhost:389");
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "cn=ldap-admin,dc=test-ldap,dc=net");
    env.put(Context.SECURITY_CREDENTIALS, "inconnu");

    DirContext dirContext;

    try {
      dirContext = new InitialDirContext(env);
      dirContext.close();
    } catch (NamingException e) {
      System.err.println("Erreur lors de l'acces au serveur LDAP" + e);
      e.printStackTrace();
    }
  }
}
```

Exemple d'utilisation LDAP (2/2)

- Comme l'objet InitialDircontext encapsule la connexion vers l'annuaire, il est nécessaire de fermer cette connexion dès que celle ci n'est plus utilisée en faisant appel à la méthode close().
- La plupart des méthodes de la classe InitialDirContext peuvent lever une exception de type `NamingException`.
- Si les informations de connexion au serveur sont erronées alors une exception de type `javax.naming.CommunicationException`.
- Si les informations fournie pour l'authentification sont erronées alors une exception de type javax.naming.AuthenticationException est levée avec le message "[LDAP : error code 49 - Invalid Credentials]".
- Á partir d'une instance de DirContext, il est possible d'accéder et de réaliser des opérations dans l'annuaire.

Manipulation des attributs en Java

- Pour manipuler les attributs d'un objet, deux interfaces existent :
  - Attributes : qui encapsulent les différents attributs d'un objet,
  - Attribut qui encapsule la valeur d'un attribut.

```
Attributes attributs = dirContext.getAttributes("cn=Duvallet,dc=test-ldap,dc=net");
Attributs attribut = (Attribut) attributs.get("description") ;
System.out.println("Description : " + attribut.get());
```

- Deux classes implémentent respectivement ces deux interfaces : BasicAttributes et BasicAttribut.
- Il est possible d'instancier une liste d'attributs par exemple pour les associer à un nouvel objet ajoutés dans l'annuaire.

```
Attributes attributes = new BasicAttributes(true);
Attribute attribut = new BasicAttribute("telephoneNumber");
attribut.add("99.99.99.99.99");
attributes.put(attribut);
```

Utilisation d'objets en Java

- La possibilité de stocker des objets Java dans un annuaire LDAP offre plusieurs intérêts :
  - Stocker des objets accessibles par plusieurs applications
  - Stocker des objets entre plusieurs exécutions d'une même application
  - Stocker des objets pour échanger des données entre plusieurs applications
- Á partir d'un objet de type contexte, il suffit de faire appel à la méthode `bind()` qui attend en paramètre un nom d'objet et un objet.
- Cette méthode va ajouter une entrée dans l'annuaire qui va associer le nom de l'objet à l'objet fourni en paramètre.
- La méthode `lookup()` d'un objet de type context permet d'obtenir un objet Java stockée dans l'annuaire à partir de son nom.

Stockage d'objets en Java (1/2)

- La plupart des annuaires permettent le stockage d'objets Java :
  - sous réserve que l'annuaire le propose,
  - et que le schéma adéquat soit utilisé dans la configuration du serveur,
  - ce qui n'est généralement pas le cas par défaut.
- Le stockage se fait en utilisant la méthode `bind()` du contexte.

```
try {
    dirContext = new InitialDirContext(env);
    MonObjet objet = new MonObjet("valeur1","valeur2");

    dirContext.bind("cn=monobject,dc=test-ldap,dc=net", objet);
    dirContext.close();
} catch (NamingException e) {
    System.err.println("Erreur lors de l'acces au serveur LDAP" + e);
    e.printStackTrace();
}
```

Stockage d'objets en Java (2/2)

- Les objets Java peuvent être stockés de différentes manières selon le serveur :
  - Stockage des objets eux mêmes sous la forme sérialisée.
  - Stockage d'une référence mémoire vers l'objet Java : cette référence est encapsulée dans un objet de type `java.naming.Reference`.
  - Stockage des champs de l'objet sous la forme d'attributs : l'objet ainsi stocké doit obligatoirement implémenter l'interface `DirContext`.
- L'implémentation des toutes ces méthodes est laissée libre au serveur mais au moins une doit être proposée.
- Pour le stockage sous la forme sérialisée, il est nécessaire que l'objet stocké implémente l'interface `java.io.Serilizable`.
- C'est la solution la plus facile à mettre en œuvre et celle que nous avons utilisé.

L'objet sérialisé

```
import  java.io.Serializable;

public class MonObjet implements Serializable {

    private String champ1;
    private String champ2;

    public MonObjet() {
        super();
    }

    public MonObjet(String champ1, String champ2) {
        super();
        this.champ1 = champ1;
        this.champ2 = champ2;
    }

    public String getChamp1() {
        return champ1;
    }

    public void setChamp1(String champ1) {
        this.champ1 = champ1;
    }

    public String getChamp2() {
        return champ2;
    }

    public void setChamp2(String champ2) {
        this.champ2 = champ2;
    }
}
```