Exercice3

Le but de cet exercice est d'illustrer le module spring-data-ldap. Pour ce faire nous utilisons un mapping ODM (Spring LDAP Object Directory Mapping). ODM offre la possibilité d'utiliser des annotations pour mapper des annuaires LDAP à des objets Java.

Etape 1 : création du projet

A partir d'Eclipse, créer un projet comme précédemment mais le nommé : ex3-spring-data afin d'illustrer les concepts LDAP au sein de spring-data

Ajouter les dépendances sur le starter spring ldap et sur le driver du server ldap embarqué.

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-ldap</artifactId>
	</dependency>
	<dependency>
		<groupId>com.unboundid</groupId>
		<artifactId>unboundid-ldapsdk</artifactId>
	</dependency>
</dependencies>
```

Etape 2 : Configuration du serveur LDAP

Nous utilisons Spring Boot pour créer et configurer notre serveur LDAP intégré. Les propriétés suivantes créent un serveur LDAP s'exécutant sur le port 12345 et renseignent le serveur LDAP à l'aide du fichier `application.yml` qui se trouve dans le CLASSPATH.

```
# Spring LDAP CRUD Operations Binding and Unbinding Example

spring:
  ldap:

    # Spring LDAP
    #
    # Dans cet exercice, nous utilisons un serveur LDAP intégré. Si vous en
    # utilisez un réel, vous pouvez configurer les paramètres ici.
    #
    # urls: ldap://localhost:12345
    # base: dc=data,dc=com
    # username: uid=admin
    # password: secret

    # Embedded Spring LDAP
    embedded:
      base-dn: dc=data,dc=com
      credential:
        username: uid=admin
        password: secret
      ldif: classpath:schema.ldif
      port: 12345
      validation:
        enabled: false
```

Les serveurs LDAP sont renseignés à l'aide du fichier `schema.ldif` suivant, présent dans le CLASSPATH. LDIF : LDAP Data Interchange Format est le format de fichier permettant le chargement et la mise à jour de données dans un annuaire LDAP.

Notes :

- Noter que le séparateur entre 2 objets est une ligne vide.
- Les classes d'objets (objectClass) : pour le moment, une classe définit les différents attributs qu'un objet peut ou doit posséder.

- Le DN est le « distinguished name », à savoir le nom de l'objet dans l'annuaire. C'est ce nom qui permet de retrouver de façon unique un objet dans l'arbre des objets. Un objet possède donc toujours un DN, qui reprend son RDN (relative dn). Ici, le RDN de dc=data, dc=com est dc=data et cn=John Deuf pour le 1er people.

Remplir le serveur LDAP à l'aide du fichier `schema.ldif` suivant.

```
dn: dc=data,dc=com
objectclass: top
objectclass: domain
objectclass: extensibleObject
dc: data

# Organizational Units
dn: ou=groups,dc=data,dc=com
objectclass: top
objectclass: organizationalUnit
ou: groups

dn: ou=people,dc=data,dc=com
objectclass: top
objectclass: organizationalUnit
ou: people

# Create People
dn: uid=john,ou=people,dc=data,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: John Doe
sn: John
uid: john
password: secret

dn: uid=jihn,ou=people,dc=data,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Jihn Die
sn: Jihn
uid: jihn
password: secret

dn: uid=jahn,ou=people,dc=data,dc=com
objectclass: top
```

```
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Jahn Dae
sn: Jahn
uid: jahn
password: secret

# Create Groups
dn: cn=developers,ou=groups,dc=data,dc=com
objectclass: top
objectclass: groupOfUniqueNames
cn: developers
ou: developer
uniqueMember: uid=john,ou=people,dc=data,dc=com
uniqueMember: uid=jihn,ou=people,dc=data,dc=com

dn: cn=managers,ou=groups,dc=data,dc=com
objectclass: top
objectclass: groupOfUniqueNames
cn: managers
ou: manager
uniqueMember: uid=jahn,ou=people,dc=data,dc=com
```

Etape 3 : Création du mapping LDAP des nœuds de base « people »

Nous rappelons les annotations dédiées pour LDAP.

- `@Entry` - Annotation au niveau de la classe indiquant les définitions de `objectClass` auxquelles l'entité est mappée. (Champs obligatoires)
- `@Id` - Indique le nom distinctif de cette entité (DN); le champ déclarant cet attribut doit être un dérivé de la classe `javax.naming.Name`. (Champs obligatoires)
- `@Attribute` - Indique le mapping d'un attribut de l'annuaire sur le champ de la classe d'objet.
- `@DnAttribute` - Indique le mapping d'un attribut `dn` sur le champ de la classe d'objet.
- `@Transient` - Indique que le champ n'est pas persistant et doit être ignoré par `OdmManager`.

Pour qu'une entrée d'annuaire soit considérée en tant que correspondance avec l'entité gérée, toutes les objets déclarés par une entrée de l'annuaire doivent avoir une correspondance dans l'annotation `@Entry`.

Par exemple: supposons que l'arborescence LDAP contient des entrées dont les classes sont `inetOrgPerson, organizationPerson, person, top.`

Si nous souhaitons uniquement modifier les attributs définis dans objectclass person, l'annotation `@Entry` doit être `@Entry (objectClasses = {"person", "top"})`. Toutefois, si nous souhaitons gérer les attributs définis dans la classe d'objets `inetOrgPerson`, nous devons utiliser l'intégralité soit: `@Entry (objectClasses = {"inetOrgPerson", "organizationPerson", "person", "top"}).`

Il est demandé de créer, mettre à jour et supprimer certains utilisateurs. Nous utilisons la classe `ex3.data.Person` pour représenter une entrée LDAP de personne. Cette classe est associée avec des annotations ODM spécifiques, mappant l'objet à l'homologue LDAP. Elle dispose des attributs

- `Name dn` annoté `@Id`
- `String uid` annoté `@DnAttribute(value="uid")`
- `String fullName` annoté `@Attribute(name="cn")`
- `String lastName` annoté `@Attribute(name="sn")`
- `String group` annoté `@DnAttribute(value="ou")` et `@Transient`

Elle dispose des méthodes suivantes :

- Un constructeur par défaut et un constructeur qui initialise les attributs à partir de ses paramètres
- Des getters et setters
- Une méthode `toString`

En utilisant des annotations, nos requêtes sont beaucoup simplifiées. Créer l'interface `ex3.data.PersonRepository` qui gère les opérations de base CRUD effectuées sur le serveur LDAP. Ces opérations incluent, mais ne sont pas limitées à: Créer, Lire, Mettre à jour et Supprimer. Cette interface hérite de `LdapRepository`. Il est demandé d'ajouter des méthodes de recherche personnalisées :

- Pour rechercher une `Person` à partir de son `lastname`,
- Pour rechercher une `Person` à partir de son `uid`,

Etape 4 : Création du mapping LDAP des nœuds supérieurs « groups »

Nous examinons maintenant la gestion de nos groupes. Nous pouvons mapper l'objet Group à l'homologue LDAP à l'aide de Spring ODM Annotations.

Il est demandé de créer une classe `ex3.data.Group` annotée `@Entity` avec les attributs suivants

- `Name dn` annoté `@Id`
- `String name` annoté `@Attribute(name="cn")` et `@DnAttribute("cn")`
- `Set members` annoté `@Attribute(name="uniqueMember")`

Elle dispose des méthodes suivantes :

- Un constructeur par défaut et un constructeur qui initialise les attributs à partir de ses paramètres
- Des getters et setters
- Une méthode `toString`

Créer l'interface `ex3.data.GroupRepository` qui gère les opérations de base CRUD sur les groups. Cette interface hérite de `LdapRepository`. Il est demandé d'ajouter des méthodes de recherche personnalisées :

- Rechercher un Group à partir d'un nom de propriété et de sa valeur

Etape 5 : Préparation de l'Application Context

Nous pouvons configurer Spring Data LDAP à l'aide de classes @Configuration basées sur Java ou d'un espace de noms XML. Configurons le référentiel à l'aide de l'approche Java.

Il est demandé de :

- Créer une classe `ex3.AppConfig` annotée `@Configuration` et `@EnableLdapRepositories` à laquelle on indique le package où sont déclarés les Repositories.
- Créer une classe `ex3.AppMain` annotée `@SpringBootApplciation`. Avec 2 attributs injectés par type :
    - `PersonRepository personRepository`
    - `GroupRepository groupRepository`
- Une fois l'application initialisée, nous exécutons certaines opérations sur le serveur LDAP pour illustrer l'usage des repositories précédents.
  Dans ce but, créer une méthode init annotée @PostConstruct

```
List<Person> persons = personRepository.findAll();
log.info("persons: " + persons);

Person olivier = new Person("Olivier Sips", "Sips");
personRepository.create(olivier);

Person john = personRepository.findByUid("john");
personRepository.delete(john);
```

Etape 6 : Test

Lancer l'application pour visualiser les résultats des opérations sur l'annuaire LDAP.

Exercice4

Le but de cet exercice est d'illustrer le module spring-data-mongoDB.

Etape 1 : création du projet

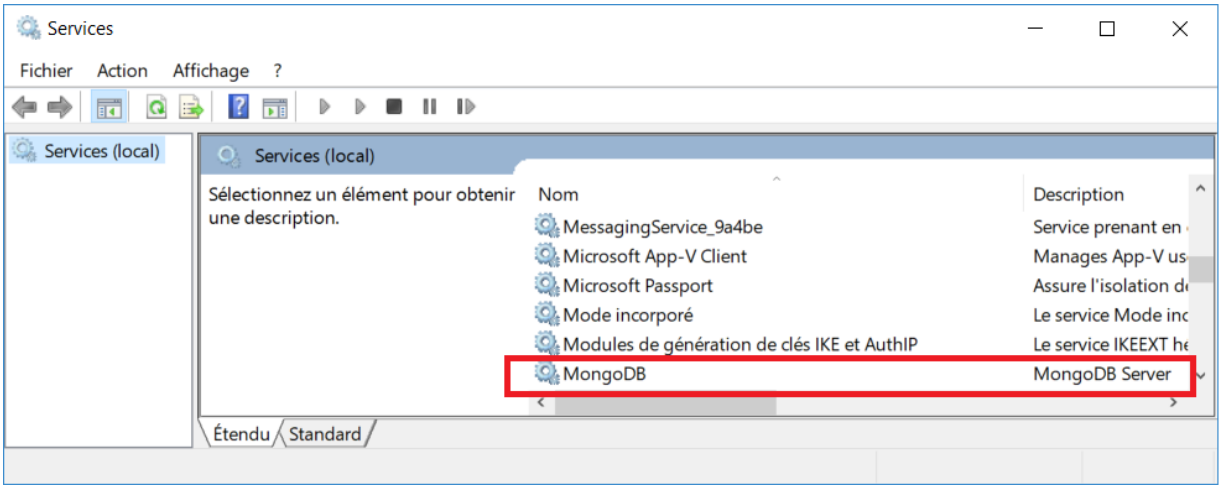
A partir d'Eclipse, créer un projet comme précédemment mais le nommé : ex4-spring-data afin d'illustrer les concepts MongoDB au sein de spring-data

Ajouter les dépendances sur le starter spring mongoDB et sur le driver du server Mongo.

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-mongodb</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>
</dependencies>
```

Etape 2 : Configuration de MongoDB

Valider que le service `MongoDB` est actif par l'ouverture de la fenêtre de services sous Windows :



Sinon installer le depuis http://www.mongodb.org/downloads et le démarrer

Au niveau du projet Maven, il est indispensable de définir les propriétés de connexion dans un fichier de propriétés nommé `application.properties` placé dans le répertoire `src\main\resources`.

```
	#mongodb
	spring.data.mongodb.host=localhost
	spring.data.mongodb.port=27017
	spring.data.mongodb.database=ex4
	#logging
	logging.level.org.springframework.data=debug
	logging.level.=error
```

Si la base n'existe pas déjà, elle sera créée.

Etape 3 : Création du mapping

Pour l'exemple créer une classe `ex4.data.Domain` qui joue le rôle d'entité, elle sera annoté @Document(collection = "domain"), cette classe dispose de 3 attributs :

- `Long id` annoté `@id`
- `String domain` annoté `@Indexed`
- `Boolean displayAds` sans annotation
- Cette classe dispose 'un constructeur ainsi que des getters et setters et une méthode `toString`.

Ensuite créer une interface `DomainRepository` qui hérite de `MongoRepository` et qui permet de gérer des `Domain` en base. Cette classe permet de définit des méthodes de recherche personnalisée telle que :

- Rechercher le premier `Domain` à partir de l'attribut `domain`
- Rechercher un `Domain` à partir du `domain` et du `displayAds`
- Faire une méthode de recherche personnalisée d'un `Domain` annoté `@Query` pour ajouter sa requête de recherche JSON sur l'attribut `domain`.
- Faire une méthode de recherche personnalisée d'une liste de `Domain` à partir d'une expression régulière sur le nom de `domain`

Pour créer une méthode personnalisée pour `DomainRepository`, on doit créer l'implémentation dans un autre fichier et le faire en sorte que `DomainRepository` l'étende.

Créer une interface `DomainRepositoryCustom` et ajouter une méthode personnalisée «update» `domain` et `displayAds`.

Créer une classe `DomainRepositoryImpl` qui implémente `DomainRepositoryCustom` ayant un attribut injecté par Spring de type `MongoTemplate`

Modifier l'interface `DomainRepository` afin qu'elle hérite aussi de `DomainRepositoryCustom`.

Etape 4 : Exécution

Créer une classe `ex4.AppMain` annotée `@SpringBootApplication` auquel on ajoute une méthode annoté `@PostConstruct` pour tester le mapping.

```
	Domain obj = domainRepository.findOne(7L);
	System.out.println(obj);

	Domain obj2 = domainRepository.findFirstByDomain("dawan.fr");
	System.out.println(obj2);

	int n = domainRepository.updateDomain("dawan.fr", true);
	System.out.println("Number of records updated : " + n);
```