



École Doctorale en Sciences Economique, Gestion et Informatique
Faculté des Sciences Économiques et de Gestion de Sfax
Département Informatique

Auditoire

Mastère Professionnel Audit et Sécurité Informatique

Architectures des Systèmes d'Information (JavaEE)

Enseignante

AÏDA KHEMAKHEM

Année Universitaire

2019 – 2020

Chapitre 2 :

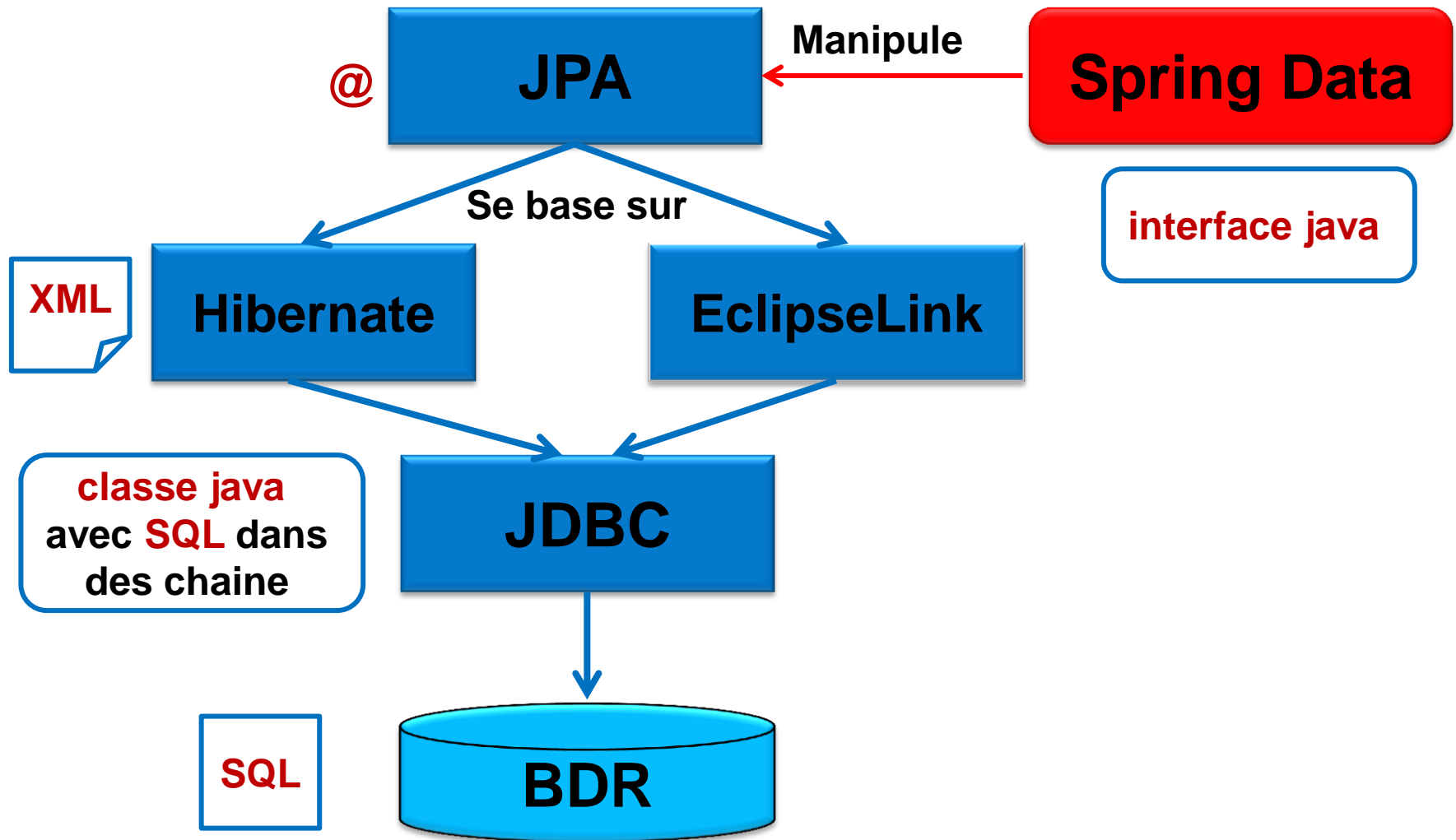
Spring Data JPA

- I. Gérer les entités EJB
- II. Spring Data JPA : interface JpaRepository
- III. SpringApplication
- IV. ApplicationContext
- V. Les principes de base : Spring Data - JPA

I. Gérer les entités EJB

- Pour gérer les entités JPA (@Entity), Spring Boot offre un module Spring Data qui offre :
 - des interfaces génériques : JpaRepository, CrudRepository...
 - Et des implémentations génériques pour **CRUD** (**Create, Read, Update, Delete**) des entités JPA
- Il suffit de créer une interface qui hérite de l'interface JpaRepository pour hériter toutes les méthodes classiques qui permettent de gérer les entités JPA
- En cas de besoin, vous avez la possibilité d'ajouter d'autres méthodes en les déclarants à l'intérieur de l'interface JpaRepository, sans avoir besoin de les implémenter, Spring Data le fera à votre place

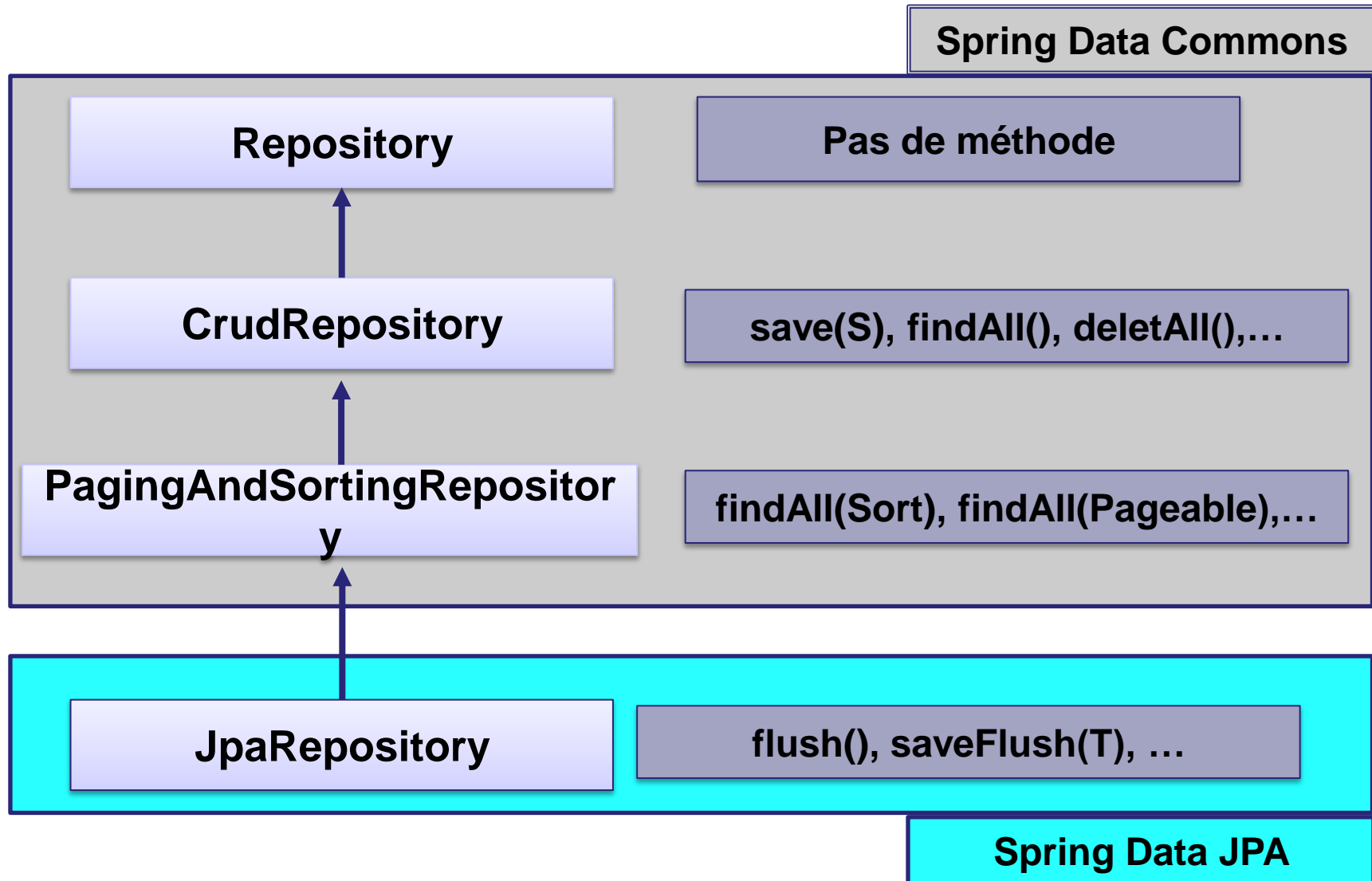
I.1. Relation entre les composants



I.2. Spring Data

- Spring Data offre une couche d'abstraction supplémentaire par rapport à JPA
- Il facilite l'écriture des couches d'accès aux BD
- Il se charge de **l'implémentation** des fonctionnalités les **plus courantes** des DAO
- On se concentre sur l'essentiel : l'écriture des requêtes

I.3. Les différentes interfaces de Spring Data



II. Spring Data JPA : interface **JpaRepository**

- Le modèle de programmation se base sur les

```
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>{ }
```

- Sans implementer les méthodes de l'interface
- Requête fournit par Spring Data

```
ApplicationContext ctx ;  
  
ctx=SpringApplication.run(GestionEtudiantsApplication.class, args);  
EtudiantDAO dao = ctx.getBean(EtudiantDAO.class);
```

- Requête dérivée du nom de la méthode

```
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>  
{ public ArrayList<Etudiant>findAll();  
  public ArrayList<Etudiant>findByNom(String nom); }
```


III. SpringApplication

- SpringApplication charge un context spring sous forme d'un objet `ApplicationContext` à partir de la méthode `run()`
- Le conteneur de Spring assure :
 - la configuration selon le fichier `application.properties`
 - La création des instances selon le scanner des fichiers (`@Entity`, `@Controller.....`)
 - Le chargement et la gestion des objets requis

```
@SpringBootApplication
public class GestionEtudiantsApplication {

    public static void main(String[] args) {
        ApplicationContext ctx;

        ctx =SpringApplication.run(GestionEtudiantsApplication.class, args);
    }
}
```



IV. ApplicationContext

- Le chargement de l'ApplicationContext est assuré par `SpringApplication.run()` qui assure
 - La création des classes qui implémentent les interfaces qui héritent de `JpaRepository`
 - La création des requêtes (notamment la traduction du nom d'une méthode-requête en requête JPQL)
- Les méthodes **`getBean(JpaRepository.class)`** : retourne une instance de la classe qui implémente l'interface en paramètre

```
EtudiantDAO dao = ctx.getBean(EtudiantDAO.class);
```



interface hérite de JpaRepository

V. Les principes de base : Spring Data - JPA

- Dans la couche DAO définit **une interface** pour chaque entité de la couche métier
- Cette interface hérite d'une interface fournie par Spring Data (JpaRepository)
- Cette interface est générique, elle a besoin de l'entité et le type de l'identifiant

```
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>
{
}
```

Nom de l'entité qu'on manipule

Type de l'identifiant de l'entité

V.1. Les principales méthodes

■ Les méthodes proposées par JpaRepository

- save() : pour sauvegarder l'objet en paramètre dans la BD
- delete() : pour supprimer l'objet en paramètre dans la BD
- findAll() : retourne une List qui regroupe toutes les données de la BD sous forme d'objet

```
@SpringBootApplication
public class GestionEtudiantsApplication {
    public static void main(String[] args) {
        ApplicationContext ctx ;
        ctx=SpringApplication.run(GestionEtudiantsApplication.class, args);
        EtudiantDAO dao = ctx.getBean(EtudiantDAO.class);
        dao.save(new Etudiant("khem", "Mohamed", new Date()));
        dao.save(new Etudiant("Sallemi", "Ahmed", new Date()));
        System.out.println(dao.findAll());
    }
}
```

V.2. Les méthodes pour les requêtes standards

- Ecrire la signature (nom) de la méthode dans l'interface
- Ce nom est composé par des propriétés (attributs) et des mots-clés mentionnés

```
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>
{
    public ArrayList<Etudiant>findAll();
    public ArrayList<Etudiant>findByNom(String nom);
    public List<Etudiant> findByNomAndPrenom(String nom, String prenom);
}
```

- A partir de nom de la méthode Spring Data assure la création d'une requête (CRUD ou de recherche)

Les mots clés pour les noms de méthode

Keyword	Sample	JPQL snippet
And	<code>findByLastnameAndFirstname</code>	<code>... where x.lastname = ?1 and x.firstname = ?2</code>
Or	<code>findByLastnameOrFirstname</code>	<code>... where x.lastname = ?1 or x.firstname = ?2</code>
Between	<code>findByStartDateBetween</code>	<code>... where x.startDate between 1? and ?2</code>
LessThan	<code>findByAgeLessThan</code>	<code>... where x.age < ?1</code>
GreaterThan	<code>findByAgeGreaterThan</code>	<code>... where x.age > ?1</code>
After	<code>findByStartDateAfter</code>	<code>... where x.startDate > ?1</code>
Before	<code>findByStartDateBefore</code>	<code>... where x.startDate < ?1</code>
IsNull	<code>findByAgeIsNull</code>	<code>... where x.age is null</code>
IsNotNull,NotNull	<code>findByAge(Is)NotNull</code>	<code>... where x.age not null</code>
Like	<code>findByFirstnameLike</code>	<code>... where x.firstname like ?1</code>
NotLike	<code>findByFirstnameNotLike</code>	<code>... where x.firstname not like ?1</code>
StartingWith	<code>findByFirstnameStartingWith</code>	<code>... where x.firstname like ?1 (parameter bound with appended %)</code>
EndingWith	<code>findByFirstnameEndingWith</code>	<code>... where x.firstname like ?1 (parameter bound with prepended %)</code>
Containing	<code>findByFirstnameContaining</code>	<code>... where x.firstname like ?1 (parameter bound wrapped in %)</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>	<code>... where x.age = ?1 order by x.lastname desc</code>
Not	<code>findByLastnameNot</code>	<code>... where x.lastname <> ?1</code>
In	<code>findByAgeIn(Collection<Age> ages)</code>	<code>... where x.age in ?1</code>
NotIn	<code>findByAgeNotIn(Collection<Age> age)</code>	<code>... where x.age not in ?1</code>
True	<code>findByActiveTrue()</code>	<code>... where x.active = true</code>
False	<code>findByActiveFalse()</code>	<code>... where x.active = false</code>

Exemple de méthodes

```
public ArrayList<Etudiant>findByNom(String nom);
```

- **findBy** : indique que l'opération à exécuter est un SELECT ;
- **Nom** : fournit le nom de la propriété sur laquelle le SELECT s'applique et il sera utiliser dans la clause WHERE
- La valeur à appliquer à la condition est, quant à elle, définie par le paramètre **nom**

```
public List<Etudiant> findByNomAndPrenom(String nom, String prenom);
```

- **findBy** : indique que l'opération à exécuter est un SELECT ;
- **NomAndPrenom** : fournit les noms des propriétés qui seront utiliser dans la clause WHERE avec And
- Les valeurs à appliquer à la condition sont définie par les paramètres **nom** et **prenom** de la méthode

V.3. Les méthodes pour les requêtes personnalisées

- Pour créer des requêtes personnalisées avec des noms de méthodes trop longues
 - Ajouter l'annotation **@Query** et donner la requête avec langage JPQL, en utilisant **?1**, **?2**,...pour les paramètres
 - Décrire la méthode en spécifiant les types des paramètres

```
import org.springframework.data.jpa.repository.Query;
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>
{ //...
    @Query("SELECT i FROM Etudiant i WHERE i.naissance <= ?1")
    List<Etudiant> findNaissanceSince(Date date);
}
```

```
@Query("SELECT i FROM Etudiant i WHERE i.naissance <= :date")
List<Etudiant> findNaissanceSinceZ(@Param("date") Date date);
```

VI. Langage JPQL

- Le langage JPQL (Java Persistence Query Language) permet de décrire ce que l'application recherche
- JPQL est un langage de requêtes adaptée à la spécification JPA
- Inspire du langage SQL et HQL (Hibernate Query Language) mais adapté aux entités JPA
- Permet de manipuler les entités JPA et pas les tables d'une base de données
- Supporte des requêtes de type **select, update et delete**

**On manipule des entités et non pas des tables.
Le nom des entités est sensible à la casse.**

VI.1. Requêtes sur les entités « objet »

- Les requêtes JPQL travaillent avec le modèle objet et pas avec le modèle relationnel %
- Les identificateurs désignent les entités et leurs propriétés et pas les tables et leurs colonnes %
- Les seules classes qui peuvent être explicitement désignées dans une requête (clause **from**) sont les entités (**@Entity**)
- Les entités sont désignées par leur nom %
- Le nom d'une entité est donné par l'attribut **name** de **@Entity** ; par défaut c'est le nom terminal (sans le nom du paquetage) de la classe

VI.2. Les clauses d'un select

- **%select** : type des objets ou valeurs renvoyées %
- **from** : où les données sont récupérées %
- **where** : sélectionne les données %
- **group by** : regroupe des données %
- **having** : sélectionne les groupes (ne peut exister sans clause group by) %
- **order by** : ordonne les données

VI.3. Clauses **where** et **having**

- Ces clauses peuvent comporter les mots-clés suivants :
 - **[NOT] LIKE, [NOT] BETWEEN, [NOT] IN**
 - **AND, OR, NOT**
 - **[NOT] EXISTS**
 - **ALL, SOME/ANY**
 - **IS [NOT] NULL**
 - **IS [NOT] EMPTY, [NOT] MEMBER OF** (pour les collections)

Exemple de requête

- **select** e **from** Employe **as** e %o
- **select** e **from** Employe e

On sélectionne des objets e de type Employe

- **select** e.nom, e.salaire **from** Employe e %o

On sélectionne le nom et le salaire l'objet e de type Employe

Exemple de requête avec jointure

- On peut faire des jointures comme avec SQL

```
select at from Auteur at  
      join at.livres v  
      where v.titre= 'java'
```

- On sélectionne des objets **at** de type **Auteur** qui ont un objet **v** d'une collection de type Livre. L'objet **v** a le titre 'java'
- Equivalent en SQL

```
"select *  
from Auteur a, Auteur_Livre av, Livre v  
where a.id = av.id and v.isbn = av.isbn and v.titre =  
'java'"
```

VII.4. Alias

- Le texte des requêtes utilise beaucoup les alias de classe

```
select at from Auteur at  
join at.livres v  
where v.titre= 'java'
```

- Les attributs des classes doivent être préfixés par les alias
- Une erreur fréquente du débutant est d'oublier les alias en préfixe

VII.5. Autres clauses

- group by
- order by
- having
- distinct
- les fonctions d'agrégation : count, avg...
- les requêtes imbriquées
- les mots-cles : all, (not) in, like, between, (not) null, (not) exists...

Remarque

- La requête JPQL précédente permet de **sélectionner un objet**
- Il est tout de même possible de sélectionner **seulement quelques attributs** d'un objet
- Dans ce cas là, le résultat est un **tableau** contenant les champs attributs sélectionnés
- Et il est impossible de modifier (ou supprimer) les valeurs de ces attributs sélectionnées

VIII. Requête paramétrées

- Un paramètre peut être désigné
 - par son numéro (**?n**), par exemple ?1, ?2, ?3.....
 - ou
 - par son nom (**:nom**) par exemple :titre, :dateNai,...
- Les paramètres sont numérotés à partir de 1 %
- Un paramètre peut être utilisé plus d'une fois dans une requête %
- L'usage des paramètres nommés est recommandé (plus lisible)

Les paramètres avec ?num ou :nom

■ Attention

- le même **nombre** de paramètre dans la requête et dans la méthode
- Respecter l'**ordre** et le **type** des paramètres

■ Paramètre désigné par ? avec **numéro** du paramètre

```
import org.springframework.data.jpa.repository.Query;
public interface EtudiantDAO extends JpaRepository<Etudiant, Long>
{
    //...
    @Query("SELECT i FROM Etudiant i WHERE naissance <= ?1")
    List<Etudiant> findNaissanceSince(Date date);
}
```

■ Paramètre désigné par : avec **nom** du paramètre

```
@Query("SELECT i FROM Etudiant i WHERE naissance <= :date")
List<Etudiant> findNaissanceSinceZ(@Param("date") Date date);
```

VII. L'implémentation de JpaRepository

Il y a deux méthodes pour l'implémentation des interfaces de **JpaRepository** proposées par Spring :

- Utiliser la méthode **getBean()** de la classe `ApplicationContext` (voir diap 9)
- L'injection de dépendance en utilisant l'annotation **@Autowired** qui peut être utiliser dans
 - des classes métiers (**@Service**)ou
 - des classes contrôleurs (**@Controller**)

Exemples d'utilisation de @Autowired

Dans la couche contrôleur : Servlet

```
@Controller
public class EtudiantController
{
    @Autowired
    private EtudiantDAO dao;

    @RequestMapping(value="/auteur")
    public String ajouterEtudiant()
    {Etudiant E2;
    E2=new Etudiant("khem","Med",
    new Date());
    dao.save(A2);
    Collection<Auteur>
    aut=dao.findAll();
    return "pageetudiant";
    }
```

Dans la couche métier

```
@Service
public class EtudiantMetier
{
    @Autowired
    private EtudiantDAO dao;

    public void ajouterEtudiant()
    {Etudiant E2;
    E2=new Etudiant("khem","Med",
    new Date());
    dao.save(A2);
    Collection<Auteur>
    aut=dao.findAll();
    }
```