

Fms Academy 2022

Formation Java Spring Angular

M10 : Spring

mohamed.el-babili@fms-ea.com

SOMMAIRE

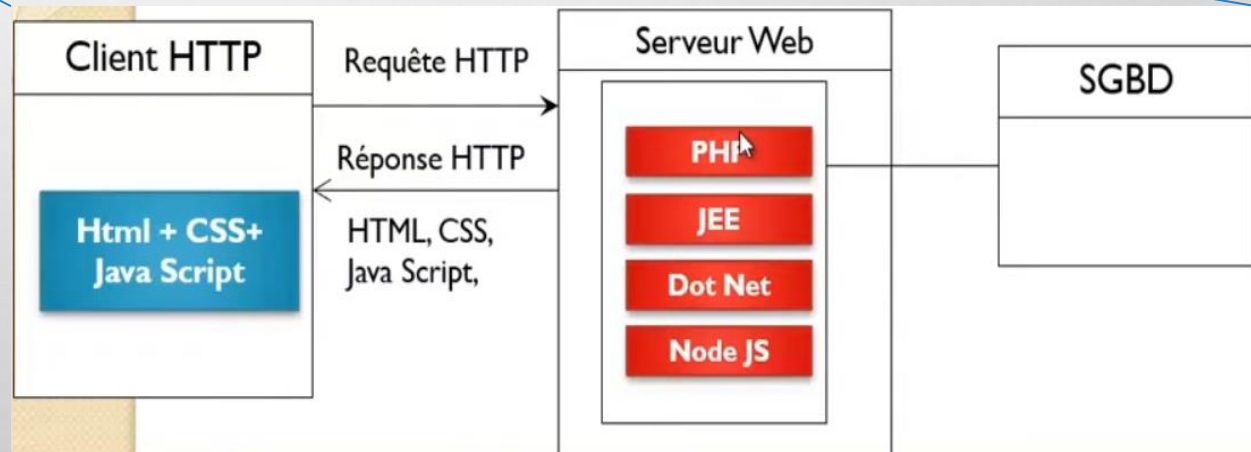
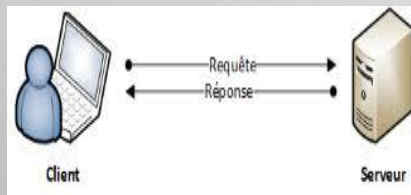
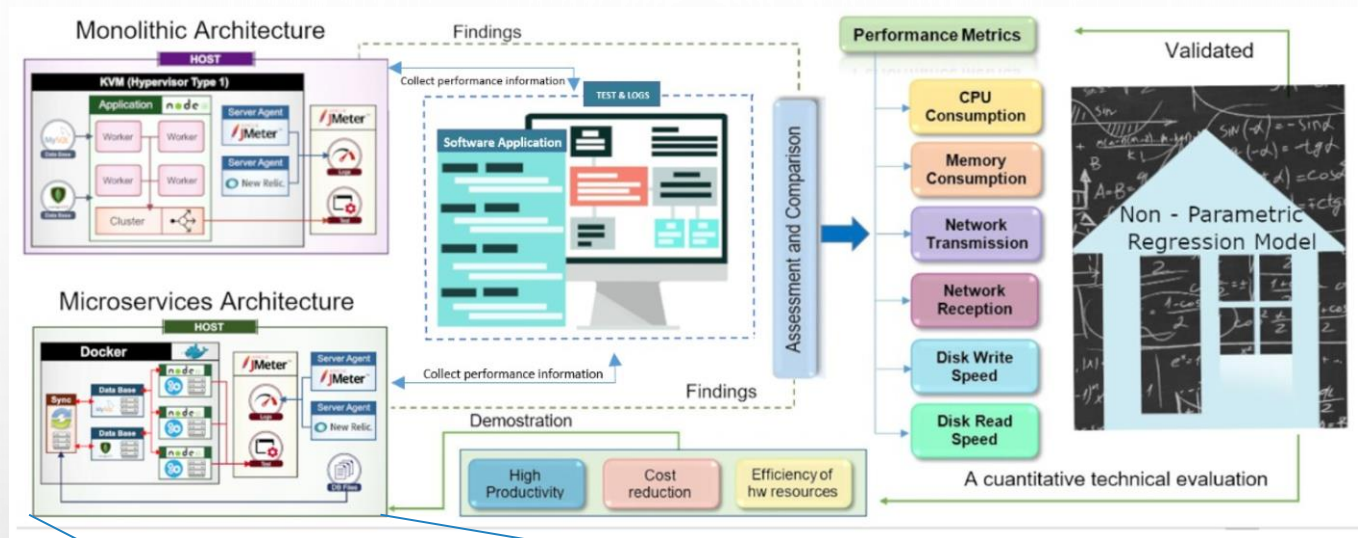
- GÉNIE LOGICIEL, INGÉNIERIE LOGICIEL
- COMPLEXITE DE L'ARCHITECTURE D'UNE APPLI WEB
- EXIGENCES OU SPÉCIFICATIONS D'UN PROJET INFORMATIQUE
- NÉCESSITÉ D'UTILISER UN FRAMEWORK
- INVERSION DE CONTRÔLE & INJECTION DES DÉPENDANCES
- MAVEN, C'EST QUOI ?
- LE FRAMEWORK SPRING
- ORM, JPA, HIBERNATE, SPRING DATA
- SPRING BOOT PAR LA PRATIQUE (steps 1 à 8)
- RESSOURCES

GÉNIE LOGICIEL

INGÉNIERIE LOGICIEL

- **DEF :** C'est une **science de génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs logiciels**. Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d'arriver à ce que des logiciels de grande taille correspondent aux attentes du client, **cad fiable avec un coût réduit et de bonnes performances tout en respectant les délais**.
- On parle même d'industrie du logiciel car en réalité, on fabrique des logiciels comme on fabrique des avions ou des voitures. En effet, tout un process doit être suivi...
- En bref, il n'existe pas une personne capable de réaliser un avion seul, ni l'ensemble des composants d'une appli aussi dans l'ingénierie logiciel, on parle de programmation par composants que les uns développent pour être réutiliser par les autres !

COMPLEXITÉ DE L'ARCHITECTURE D'UNE APPLI WEB



EXIGENCES OU SPÉCIFICATIONS D'UN PROJET INFORMATIQUE

- **FONCTIONNELLES**

→ Elles ont vocation à répondre aux besoins métier d'une entreprise !

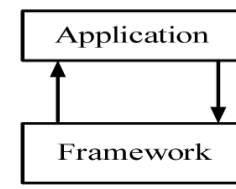
- Pour répondre à ces besoins, nous utilisons uml pour faire l'analyse et à l'aide des diagrammes de cas d'utilisation par ex, nous cherchons à déterminer quelles sont les besoins fonctionnelles (retirer, verser, consulter...)

- **TECHNIQUES**

- L'appli doit être performante (temps de réponse, pb montée en charge/architecture scalable/kubernetes, tolérance aux pannes)
- L'appli doit être facile à maintenir (doit être fermée aux modifs et ouverte aux extensions)
- L'appli doit gérer les failles de sécurité
- La persistance des données + SGBD relationnel => utilisation d'un ORM
- L'appli doit permettre la gestion des transactions
- Appli web, desktop et mobile ont en commun la couche métier ou business + DAO + SGBD aussi il faut toujours séparer la logique métier de la logique présentation (sinon l'appli sera difficile à maintenir et vouée à disparaître)

- **FINANCIÈRES** (NB DE DEVS * COUT/JH * NB JOURS)

NÉCESSITÉ D'UTILISER UN FRAMEWORK



- Il est difficile de réaliser une application qui respecte toutes les exigences avec un coût faible sans utiliser l'expérience des autres !
- Raison pour laquelle, l'utilisation de l'inversion de contrôle permet au développeur de se préoccuper uniquement des exigences fonctionnelles, c'est le framework mettant en œuvre l'IOC qui se charge des exigences techniques.
- En effet, un framework est une sorte de bibliothèque appelée par notre application qui par la suite fait des appels vers notre application, c'est en cela qu'il y a inversion de contrôle. Une des applications de l'IOC est l'ID consistant à réaliser des dépendances en amont puis préciser dans le code ou injecter ces dépendances au besoin et laisser le framework réaliser l'injection (@autowired)
- Pour utiliser l'IOC sur une architecture J2EE il existe 2 frameworks :
 - spring (très utilisé par les entreprises car conteneur léger)
 - EJB (conteneur lourd/serveur d'application J2EE)

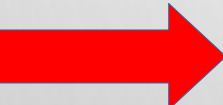
EXEMPLE D'UTILISATION DU GESTIONNAIRE DE TRANSACTION DE SPRING

```
@Override
public void transfert(long accIdSrc, long accIdDest, double amount) throws RuntimeException { //virement
    EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("Stock"); //modélise l'unité de persistance
    EntityManager entityManager = entityManagerFactory.createEntityManager(); //gère le cycle de vie des entités

    EntityTransaction transaction = entityManager.getTransaction(); //création d'une transaction
    transaction.begin(); //démarrage d'une transaction

    if(accIdSrc == accIdDest) throw new RuntimeException("vous ne pouvez retirer et verser sur le même compte !");
    try {
        withdraw(accIdSrc, amount); //retrait
        pay(accIdDest, amount); //alors versement
        transaction.commit(); //valide la transaction
    }
    catch(Exception e) {
        transaction.rollback(); //annule la transaction
        e.printStackTrace();
    }
}
```

Sans utiliser l'IOC !



```
@Transactional
public void transfert(long accIdSrc, long accIdDest, double amount) throws RuntimeException { //virement
    if(accIdSrc == accIdDest) throw new RuntimeException("vous ne pouvez retirer et verser sur le même compte !");
    withdraw(accIdSrc, amount); //retrait
    pay(accIdDest, amount); //alors versement
}
```

En utilisant l'IOC !

- On dit aussi que la séparation du code métier du code technique a été rendu possible grâce à la programmation orientée aspect (AOP/2000) permettant de greffer du code, ici développé par spring, via le mécanisme d'annotation (@transactional) lui permettant de gérer les transactions à notre place.

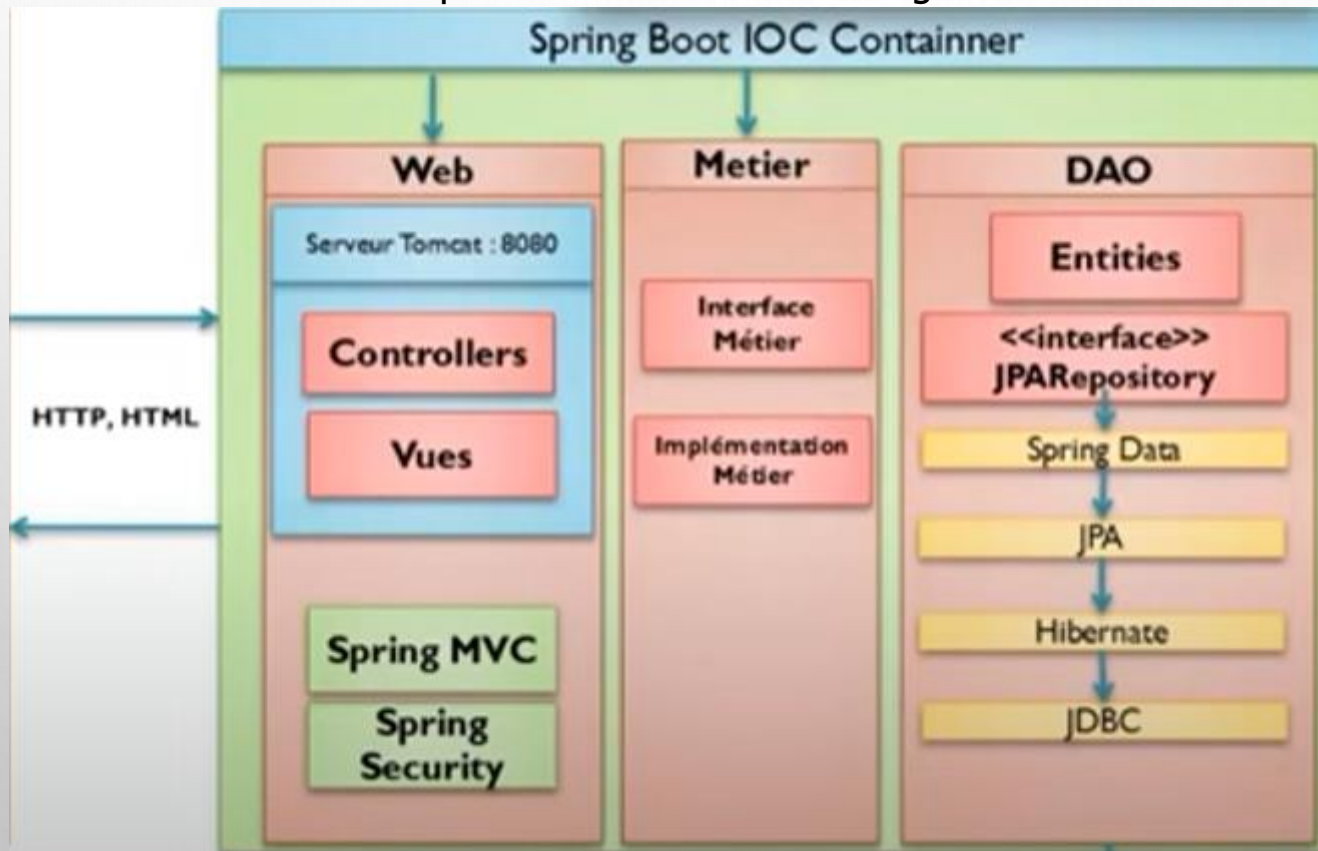
INJECTION DES DÉPENDANCES (MISE EN ŒUVRE DE L'IOC)

- Si on considère qu'une appli qui ne peut évoluer est vouée à disparaître, comment permettre à notre appli d'être facile à maintenir ? « ***fermée aux modifs et ouverte aux extensions*** » :
 - pour ce faire on utilise le couplage faible au lieu du couplage fort
- Il existe plusieurs manières de mettre en œuvre l'ID :
 - injection des dépendances par instanciation statique ou dynamique
 - injection des dépendances à l'aide de spring version xml
 - Injection des dépendances à l'aide de spring version annotations(La version xml étant plus lourde à charger du fait de l'utilisation d'un parser xml, la version annotation est plus rapide)
 - Injection des dépendances à l'aide de spring en utilisant maven
- *Nb : l'inversion de contrôle implique que spring contrôle l'app, d'ailleurs il démarre d'abord puis lance les services dont le serveur http...*

SPRING BOOT & IOC



Une fois l'application démarrée, c'est elle qui prend le contrôle et interagit avec nous 😊

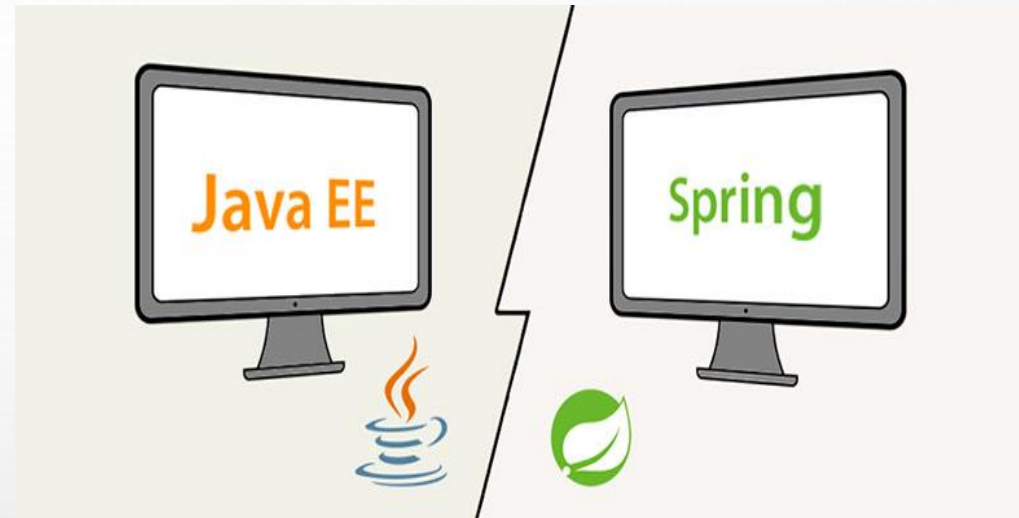
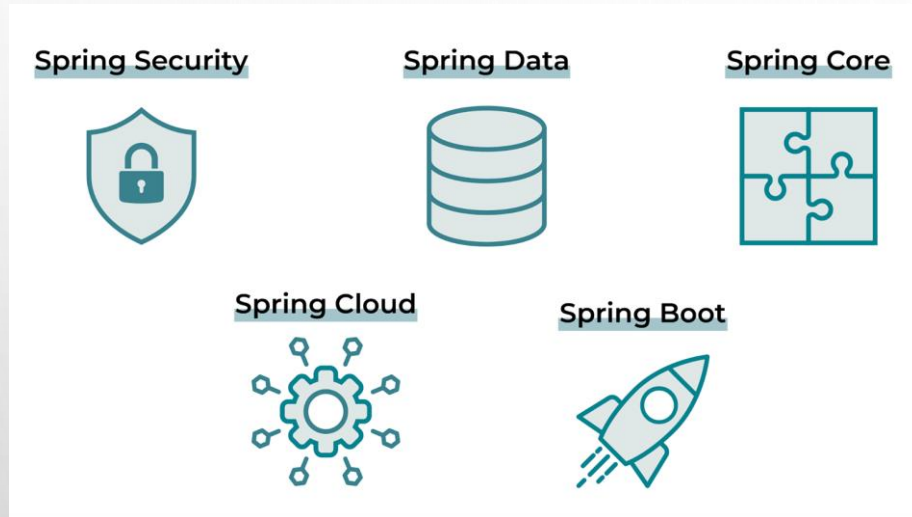


MAVEN, C'EST QUOI ?

- C'est un outil qui permet l'automatisation du processus de développement d'un projet java, il nous sera utile tout au long du cycle de vie de nos applications.
- Exemples :
 - **mvn compile** est une commande demandant à maven de compiler l'ensemble du code source de votre projet
 - **Mvn package** va générer un jar(appli desktop notamment) ou war(web app)
 - Mvn test/install/deploy/site...
- Un projet maven contient un fichier pom.Xml(project object model) notamment pour gérer l'injection des dépendances via un repository local ou distant :
 - c:\users\moam\M2\repository
- *Nb : spring peut utilisé maven ou gradle*

LE FRAMEWORK SPRING

Ce framework offre de nombreux modules permettant aux développeurs Java de se préoccuper que de leur cœur de métier : les aspects fonctionnelles



En outre, Spring est un framework nécessitant des configurations, tandis que Spring Boot est prêt à l'emploi



ORM, JPA, HIBERNATE, SPRING DATA

- 1/ Les applis ont généralement besoin d'utiliser un SGBD et dans la majorité des cas on travaille sur des bases de données relationnelles, cad que les données sont stockées sous forme de tables liées par des relations à travers le concept de clé primaire et étrangère.
- 2/ sauf que nous utilisons la POO pour réaliser nos applis.
 - Raison pour laquelle, il y a nécessité de faire le « mapping objet relationnel » consistant à faire le lien entre les 2 aspects à l'aide d'un orm, ex : hibernate
 - cette opération est réalisé dans les couches dao et entities.
 - L'idée est de créer autant de classes que vous le souhaitez avec des relations, l'orm se charge de générer automatiquement les tables en base : énorme gain de temps !

*Vous doutez
encore ?*

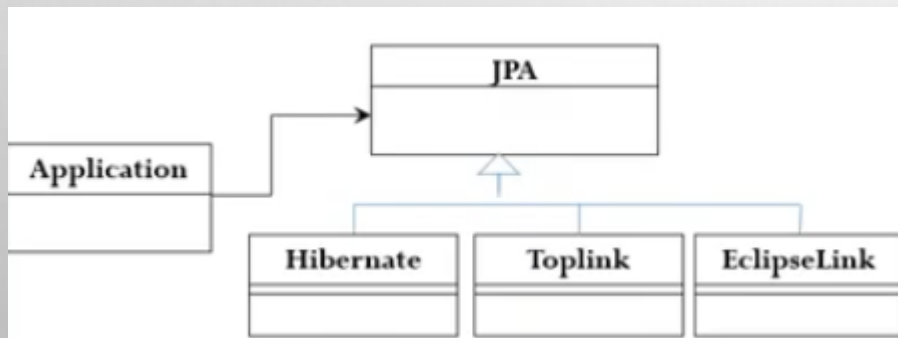
- En effet, rappelez-vous, sans orm, il faut écrire les méthodes CRUD pour chaque classe (ArticleDao)
- Dans la manière vue jusqu'à présent, nous utilisons un singleton pour gérer les connexions, sauf qu'elle présente des limites dans une appli multi threading, chaque thread ou appli doit attendre que la connexion soit libérée, la solution apportée par l'orm est un pool de connexion.
- De plus, il existe un moyen de gérer des transactions en bdd à l'aide de Spring.
- Enfin, il y a un des problèmes de sécurité avec risques d'injection sql si on utilise pas les bons objets (preparedstatement)... Nous verrons là aussi que l'utilisation combinée des frameworks spring et hibernate offre des atouts majeurs à nos applis.

ORM, JPA, HIBERNATE, SPRING DATA

- Auparavant (lorsqu'il n'existait pas de framework), pour résoudre tous ces problèmes, les développeurs étaient obligés de coder tous ces aspects.
- Sauf qu'il faut beaucoup d'expérience pour y arriver d'où les premières exp catastrophiques des jeunes ingénieurs. Raisons pour lesquelles, les entreprises demandaient 5/10 ans d'ancienneté !
- Dorénavant, les frameworks permettent de nous concentrer sur les aspects métier laissant tous les aspects technique à la charge du framework.
- Hibernate permet :
 - gain de temps
 - Garantie la portabilité si on change de sgbd car il implémente la spéc JPA
 - Efficacité et performance (débat ?)

ORM, **JPA**, **HIBERNATE**, SPRING DATA

- Il existe plusieurs orm java, tous implémentent la spécification jpa (java persistence api) crée par Sun(racheté par oracle)
- Jpa est un ensemble d'interfaces, de classes abstraites et d'annotations permettant la description du mapping objet relationnel.
- L'utilisation de jpa permet à notre appli d'être indépendante de l'Orm utilisé.



Hibernate est une implémentation de Jpa aussi notre application peut être déployé dans n'importe quel serveur d'application : Jboss, GlassFish...
Le respect de l'utilisation de la spécification garanti que cela fonctionnera !

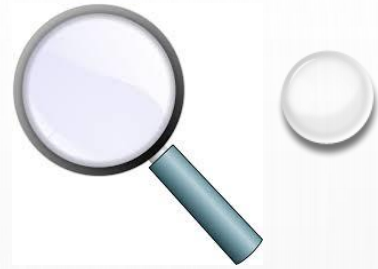
MISE EN ŒUVRE DES ANNOTATIONS JPA ?

- **@ENTITY** : La classe précédée par cette annotation a une table correspondante en base
- **@TABLE** : Permet de préciser le nom de la table car par défaut c'est le nom de la classe
- **@COLUMN** : Idem pour la colonne
- **@ID** : La propriété précédée par cette annotation sera une clé primaire
- **@GENERATEDVALUE** : Génération automatique de la clé via un auto incrément
→ Strategy = GenerationType.IDENTITY => +1
- **@TRANSIENT** : Demande de ne pas tenir compte de ce champ lors du mapping
- **@ONETOMANY, MANYTOONE** : Décrit une association type un à plusieurs et inversement
-
- ***NB : Une entité jpa est une classe serializable (peut traverser un réseaux) ou javabeen(objet java / grain de café) constitué d'attributs, un constructeur par défaut min, d'accesseurs et utilisant au minimum 2 annotations obligatoires : @entity et @id***

ORM, JPA, HIBERNATE, **SPRING DATA**

- Spring data est un module de spring permettant de simplifier l'interaction avec différents systèmes de stockage de données
- Le principe de spring data est d'éviter aux développeurs de coder les accès à ces systèmes.
- Rappelez vous lorsqu'il fallait développer une interface générique dao<t> puis l'implémenter en fonction de nos besoins articledao, userdao... et redéfinir à chaque fois chaque méthode. **Tout ce travail ne sera plus nécessaire !**
- Bien entendu, cela a un prix et exige donc de respecter certaines conditions sans quoi il existe d'autres alternatives prévues par spring data comme le codage de nos propres requêtes.

SPRING BOOT → JPA/HIBERNATE



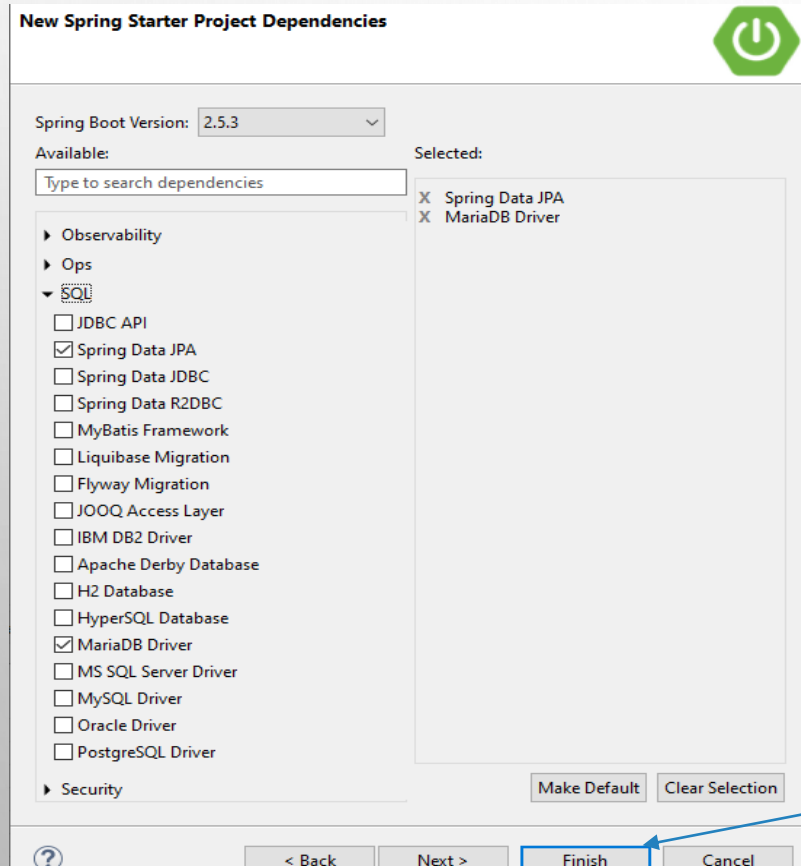
- Voir [ici](#) un exemple d'utilisation de jpa/hibernate avec maven sans spring
- Mise en œuvre des entités jpa/hibernate d'un projet avec spring boot(maven) :
 - Spring boot est une version allégée de spring permettant de faciliter la gestion des dépendances maven + gestion automatique de la configuration
 - En cochant les bonnes cases, les dépendances sont automatiquement ajoutées au fichier pom.Xml et le fichier *application.Properties* remplace le fichier *persistence.Xml* ou *config.Properties*
 - Un projet spring est constitué d'une application qui se charge de démarrer le prog(run) aussi afin de bien mettre en œuvre l'ioc, c'est bien spring qui démarre d'abord puis va lire le fichier *application.Properties*
 - A partir de là, il comprend que vous utilisez jpa et mysql par ex, il va donc automatiquement créer un objet *entitymanagerfactory* puis va lire les données sur le datasource et exécuter l'appli
 - Si tout va bien, l'application démarre et la base avec les tables sont créées

SPRING BOOT

PAR LA PRATIQUE (STEP 1)

MISE EN ŒUVRE DE TOUS LES CONCEPTS VUES

- D'ABORD, INSTALLER SPRING TOOLS (HELP/MARKETPLACE) PUIS, NEW SPRING STARTER PROJECT



New Spring Starter Project Dependencies

Spring Boot Version: 2.5.3

Available:

Type to search dependencies

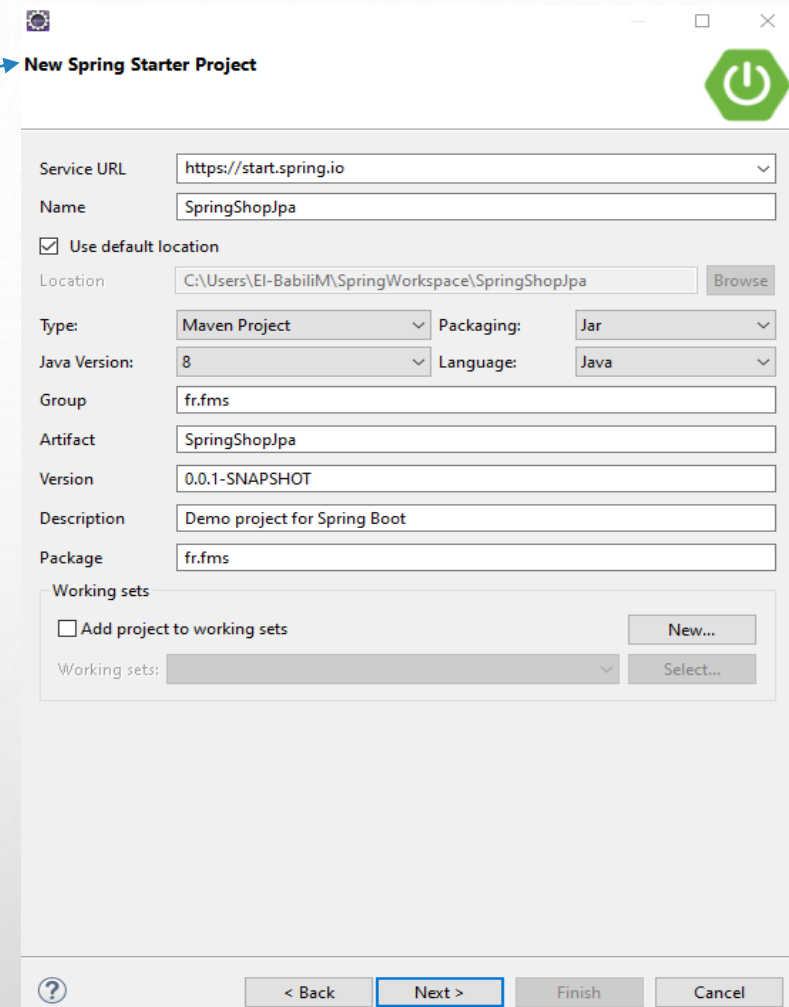
- Observability
- Ops
- SQL
 - ☐ JDBC API
 - ☒ Spring Data JPA
 - ☐ Spring Data JDBC
 - ☐ Spring Data R2DBC
 - ☐ MyBatis Framework
 - ☐ Liquibase Migration
 - ☐ Flyway Migration
 - ☐ JOOQ Access Layer
 - ☐ IBM DB2 Driver
 - ☐ Apache Derby Database
 - ☐ H2 Database
 - ☐ HyperSQL Database
 - ☒ MariaDB Driver
 - ☐ MS SQL Server Driver
 - ☐ MySQL Driver
 - ☐ Oracle Driver
 - ☐ PostgreSQL Driver
- Security

Selected:

- ☒ Spring Data JPA
- ☒ MariaDB Driver

Make Default Clear Selection

< Back Next > Finish Cancel



New Spring Starter Project

Service URL: <https://start.spring.io>

Name: SpringShopJpa

☒ Use default location

Location: C:\Users\EI-BabiliM\SpringWorkspace\SpringShopJpa Browse

Type: Maven Project Packaging: Jar

Java Version: 8 Language: Java

Group: fr.fms

Artifact: SpringShopJpa

Version: 0.0.1-SNAPSHOT

Description: Demo project for Spring Boot

Package: fr.fms

Working sets

☐ Add project to working sets New...

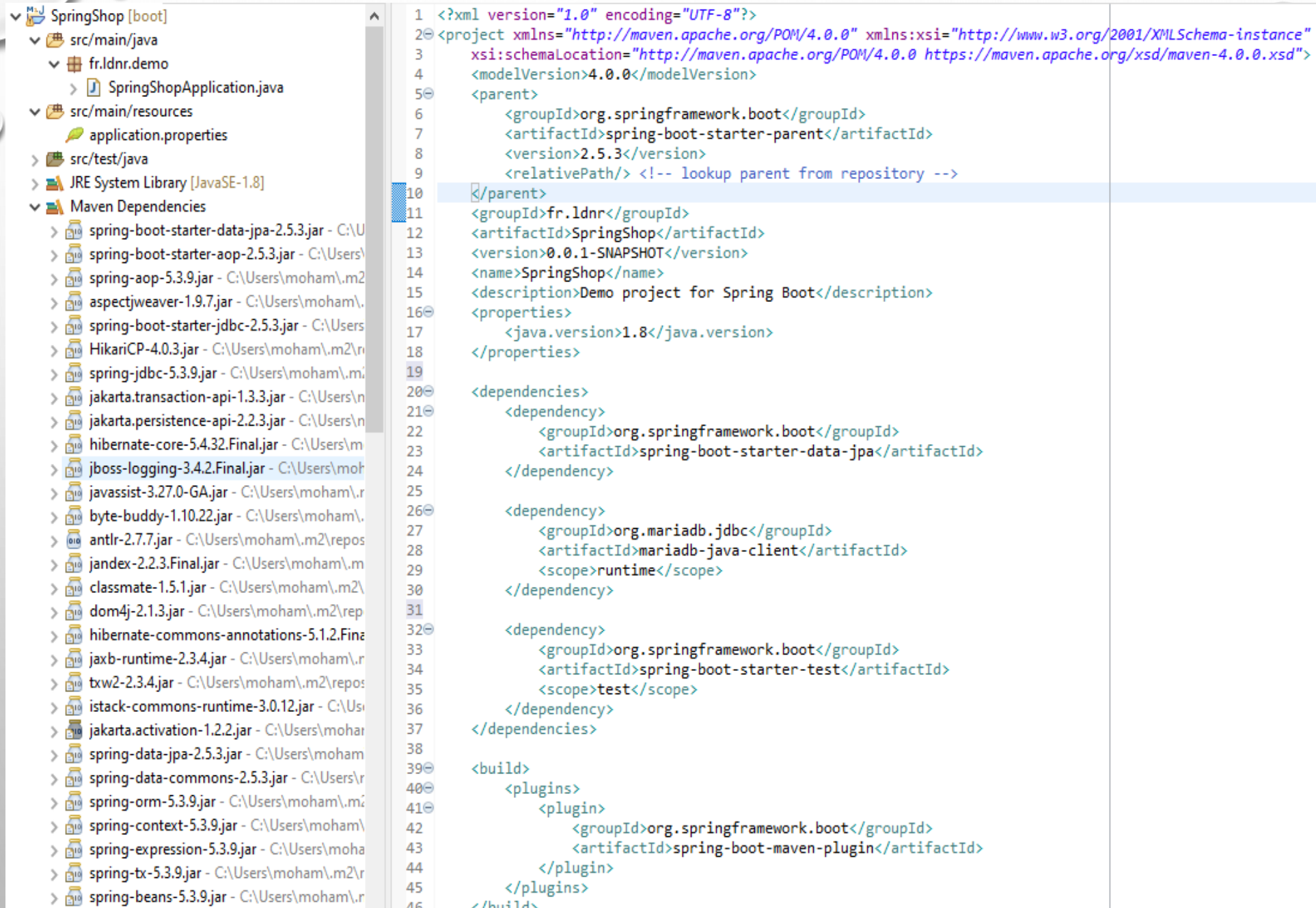
Working sets: Select...

? < Back Next > Finish Cancel

Autre manière en ligne :



SPRING BOOT PAR LA PRATIQUE (VERIF POM.XML)



The screenshot displays an IDE with two panels. The left panel shows the project structure for 'SpringShop [boot]'. The right panel shows the 'pom.xml' file.

Project Structure (Left Panel):

- SpringShop [boot]
 - src/main/java
 - fr.lidnr.demo
 - SpringShopApplication.java
 - src/main/resources
 - application.properties
 - src/test/java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - spring-boot-starter-data-jpa-2.5.3.jar - C:\U
 - spring-boot-starter-aop-2.5.3.jar - C:\Users\
 - spring-aop-5.3.9.jar - C:\Users\moham\m2
 - aspectjweaver-1.9.7.jar - C:\Users\moham\
 - spring-boot-starter-jdbc-2.5.3.jar - C:\Users
 - HikariCP-4.0.3.jar - C:\Users\moham\m2\re
 - spring-jdbc-5.3.9.jar - C:\Users\moham\m2
 - jakarta.transaction-api-1.3.3.jar - C:\Users\m
 - jakarta.persistence-api-2.2.3.jar - C:\Users\m
 - hibernate-core-5.4.32.Final.jar - C:\Users\m
 - jboss-logging-3.4.2.Final.jar - C:\Users\moh
 - javassist-3.27.0-GA.jar - C:\Users\moham\
 - byte-buddy-1.10.22.jar - C:\Users\moham\
 - antlr-2.7.7.jar - C:\Users\moham\m2\repos
 - jandex-2.2.3.Final.jar - C:\Users\moham\m
 - classmate-1.5.1.jar - C:\Users\moham\m2\
 - dom4j-2.1.3.jar - C:\Users\moham\m2\rep
 - hibernate-commons-annotations-5.1.2.Fina
 - jaxb-runtime-2.3.4.jar - C:\Users\moham\
 - txw2-2.3.4.jar - C:\Users\moham\m2\repos
 - istack-commons-runtime-3.0.12.jar - C:\Us
 - jakarta.activation-1.2.2.jar - C:\Users\moha
 - spring-data-jpa-2.5.3.jar - C:\Users\moham
 - spring-data-commons-2.5.3.jar - C:\Users\
 - spring-orm-5.3.9.jar - C:\Users\moham\m2
 - spring-context-5.3.9.jar - C:\Users\moham\
 - spring-expression-5.3.9.jar - C:\Users\moha
 - spring-tx-5.3.9.jar - C:\Users\moham\m2\
 - spring-beans-5.3.9.jar - C:\Users\moham\

POM.xml (Right Panel):

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.5.3</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>fr.lidnr</groupId>
12  <artifactId>SpringShop</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>SpringShop</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>1.8</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-data-jpa</artifactId>
23    </dependency>
24    <dependency>
25      <groupId>org.mariadb.jdbc</groupId>
26      <artifactId>mariadb-java-client</artifactId>
27      <scope>runtime</scope>
28    </dependency>
29    <dependency>
30      <groupId>org.springframework.boot</groupId>
31      <artifactId>spring-boot-starter-test</artifactId>
32      <scope>test</scope>
33    </dependency>
34  </dependencies>
35  <build>
36    <plugins>
37      <plugin>
38        <groupId>org.springframework.boot</groupId>
39        <artifactId>spring-boot-maven-plugin</artifactId>
40      </plugin>
41    </plugins>
42  </build>
```


STEP 2 : CONFIGURER L'UNITÉ DE PERSISTENCE APPLICATION.PROPERTIES

```
#Database
spring.datasource.url = jdbc:mariadb://localhost:3306/stock?createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password = fms2022
spring.datasource.driver-class-name = org.mariadb.jdbc.Driver

#Jpa-Hibernate
spring.jpa.show-sql= true
spring.jpa.hibernate.ddl-auto= update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDB53Dialect
```


STEP 3 : RÉALISATION DES ENTITÉS JPA AVEC LE MÉCANISME D'ANNOTATIONS

Project Explorer

- AdvEx4Bdd
- SpringShopJpa [boot]
 - src/main/java
 - fr.lidnr
 - SpringShopJpaApplication.java
 - fr.lidnr.dao
 - fr.lidnr.entities
 - Article.java
 - Category.java
 - src/main/resources
 - application.properties
 - src/test/java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

*Article.java

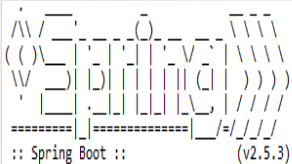
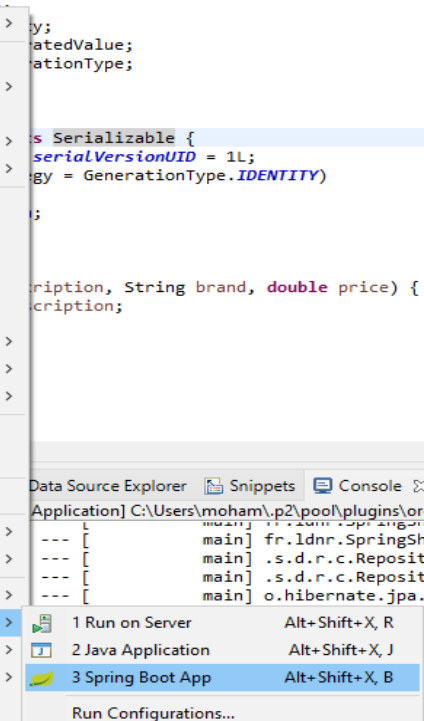
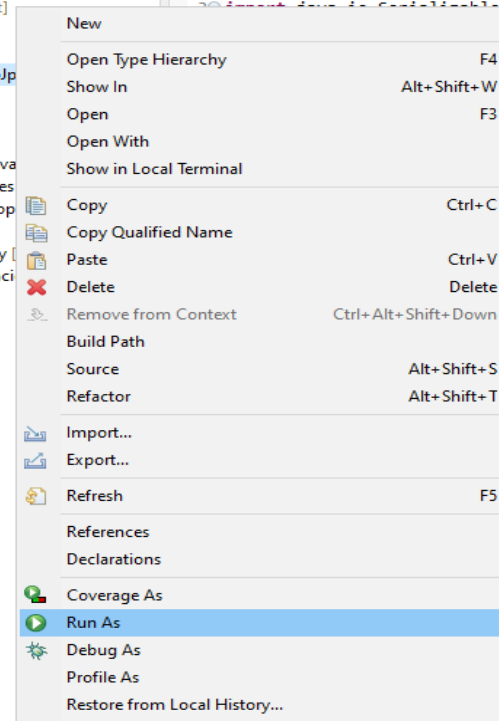
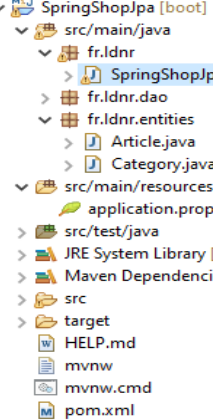
```
1 package fr.lidnr.entities;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8
9 @Entity
10 public class Article implements Serializable {
11     private static final long serialVersionUID = 1L;
12     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14     private String description;
15     private String brand;
16     private double price;
17
18     public Article(String description, String brand, double price) {
19         this.description = description;
20         this.brand = brand;
21         this.price = price;
22     }
23
24     public Long getId() {
25         return id;
26     }
27
28     public void setId(Long id) {
29         this.id = id;
30     }
31
32     public String getDescription() {
33         return description;
34     }
35 }
```

Category.java

```
1 package fr.lidnr.entities;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Category {
7     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
8     private Long id;
9     private String name;
10
11     public Category(String name) {
12         this.name = name;
13     }
14
15     public Long getId() {
16         return id;
17     }
18
19     public void setId(Long id) {
20         this.id = id;
21     }
22
23     public String getName() {
24         return name;
25     }
26
27     public void setName(String name) {
28         this.name = name;
29     }
30 }
31
32 }
```

Pour l'instant, il n'y a aucun lien entre les tables !

STEP 4 : EXÉCUTER L'APPLI ET VÉRIFIER SI LES TABLES SONT GÉNÉRÉES



```

2021-07-29 13:12:48.720 INFO 16548 --- [main] fr.lnr.SpringShopJpaApplication : Starting SpringShopJpaApplication using Java 16.0.1 on DESKTOP-OG3H4GR with PID 16548 (
2021-07-29 13:12:48.723 INFO 16548 --- [main] fr.lnr.SpringShopJpaApplication : No active profile set, falling back to default profiles: default
2021-07-29 13:12:48.783 INFO 16548 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-07-29 13:12:48.776 INFO 16548 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 76 ms. Found 2 JPA repository interfaces.
2021-07-29 13:12:48.777 INFO 16548 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2021-07-29 13:12:48.761 INFO 16548 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.32.Final
2021-07-29 13:12:48.751 INFO 16548 --- [main] o.hibernate.annotations.common.Version : HCANV000001: Hibernate Commons Annotations {5.1.2.Final}
2021-07-29 13:12:48.765 INFO 16548 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-07-29 13:12:48.765 INFO 16548 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-07-29 13:12:48.781 INFO 16548 --- [main] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.MariaDB53Dialect
2021-07-29 13:12:48.787 INFO 16548 --- [main] org.hibernate.tuple.PojoInstantiator : HH0000182: No default (no-argument) constructor for class: fr.lnr.entities.Article (c
2021-07-29 13:12:48.784 INFO 16548 --- [main] org.hibernate.tuple.PojoInstantiator : HH0000182: No default (no-argument) constructor for class: fr.lnr.entities.Category (c
Hibernate: create table article (id bigint not null auto_increment, brand varchar(255), description varchar(255), price double precision not null, primary key (id)) engine=InnoDB
Hibernate: create table category (id bigint not null auto_increment, name varchar(255), primary key (id)) engine=InnoDB
2021-07-29 13:12:48.943 INFO 16548 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.plat
2021-07-29 13:12:48.952 INFO 16548 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-07-29 13:12:49.562 INFO 16548 --- [main] fr.lnr.SpringShopJpaApplication : Started SpringShopJpaApplication in 4.38 seconds (JVM running for 5.136)
2021-07-29 13:12:49.568 INFO 16548 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2021-07-29 13:12:49.571 INFO 16548 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2021-07-29 13:12:49.577 INFO 16548 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

```

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	auto_increment
brand	varchar(255)	YES		NULL	
description	varchar(255)	YES		NULL	
price	double	NO		NULL	

4 rows in set (0.097 sec)

```

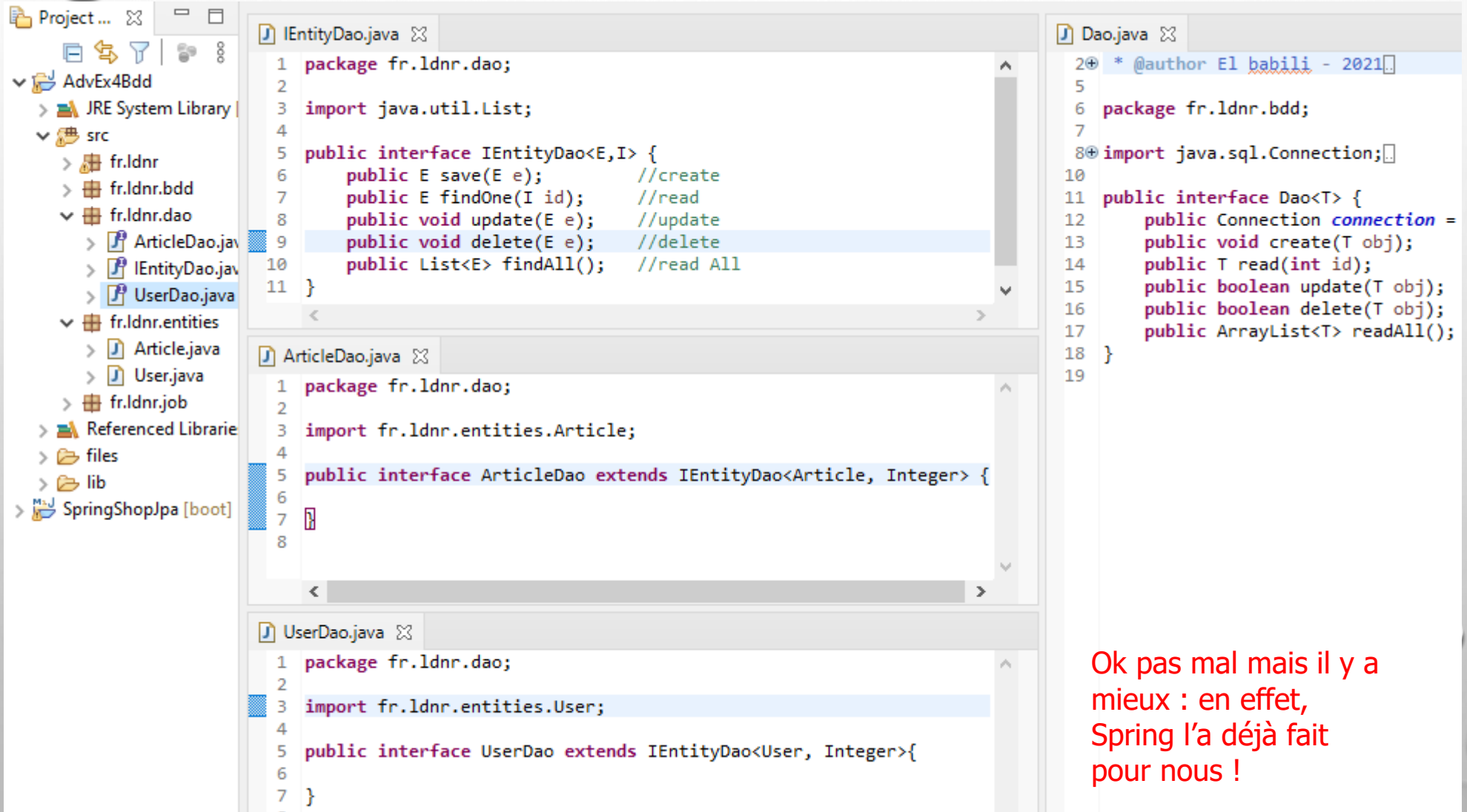
MariaDB [shop]> describe category;

```

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	

AVANT D'ALLER + LOIN, COMMENT OPTIMISER LE TRAVAIL ICI ?

En effet, imaginons que nous voudrions élargir l'interface dao<t> de sorte que nous pourrions utiliser une interface générique pour tout projet :
il faudra juste indiquer l'entité ou classe et le type d'id



The screenshot shows an IDE with three Java files open. The left pane shows a project structure with packages like `fr.ldnr`, `fr.ldnr.bdd`, `fr.ldnr.dao`, and `fr.ldnr.entities`. The main editor displays the following code:

```
IEntityDao.java
1 package fr.ldnr.dao;
2
3 import java.util.List;
4
5 public interface IEntityDao<E,I> {
6     public E save(E e);           //create
7     public E findOne(I id);       //read
8     public void update(E e);      //update
9     public void delete(E e);      //delete
10    public List<E> findAll();      //read All
11 }
```

```
ArticleDao.java
1 package fr.ldnr.dao;
2
3 import fr.ldnr.entities.Article;
4
5 public interface ArticleDao extends IEntityDao<Article, Integer> {
6
7
8 }
```

```
UserDao.java
1 package fr.ldnr.dao;
2
3 import fr.ldnr.entities.User;
4
5 public interface UserDao extends IEntityDao<User, Integer>{
6
7
8 }
```

```
Dao.java
2+ * @author El Babili - 2021
5
6 package fr.ldnr.bdd;
7
8+ import java.sql.Connection;
9
10
11 public interface Dao<T> {
12     public Connection connection =
13     public void create(T obj);
14     public T read(int id);
15     public boolean update(T obj);
16     public boolean delete(T obj);
17     public ArrayList<T> readAll();
18 }
19
```

Ok pas mal mais il y a
mieux : en effet,
Spring l'a déjà fait
pour nous !

STEP 5 : UTILISER SPRING DATA

- En effet, ce module de Spring a déjà créé une interface générique contenant plusieurs méthodes génériques fréquemment utilisées !
- De même, une implémentation générique de cette interface a été créée, elle utilise EntityManager(gestion des entités) avec les méthodes associées (findById, findAll, delete, save...)
- Il faut donc créer une interface qui hérite de [JpaRepository](#) qui aura besoin d'une entité Jpa et d'un Id associé

```
Project Explorer
└─ AdvEx4Bdd
   └─ SpringShopJpa [boot]
      └─ src/main/java
         └─ fr.ldnr
            ├── SpringShopJpaApplicat
            ├── fr.ldnr.dao
            │   ├── ArticleRepository.java
            │   └── CategoryRepository.java
            └─ fr.ldnr.entities
               ├── Article.java
               └── Category.java
      └─ src/main/resources
         └─ application.properties
      └─ src/test/java
      └─ JRE System Library [JavaSE-1.
      └─ Maven Dependencies

ArticleRepository.java
1 package fr.ldnr.dao;
2
3 import fr.ldnr.entities.Article;
4
5
6 public interface ArticleRepository extends JpaRepository<Article, Long> {
7
8 }
9

CategoryRepository.java
1 package fr.ldnr.dao;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface CategoryRepository extends JpaRepository<Category, Long> {
8
9 }
10
```

NB : Spring ici via l'IIOC nous garanti que les aspects techniques sont prêt à l'emploi, reste plus qu'à coder les aspects métier !

STEP 6 : INJECTER LES DÉPENDANCES, TESTER PUIS VÉRIFIER EN BASE

```
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.boot.CommandLineRunner;
5 import org.springframework.boot.SpringApplication;
6 import org.springframework.boot.autoconfigure.SpringBootApplication;
7
8 import fr.ldnr.dao.ArticleRepository;
9 import fr.ldnr.dao.CategoryRepository;
10 import fr.ldnr.entities.Article;
11 import fr.ldnr.entities.Category;
12
13 @SpringBootApplication
14 public class SpringShopJpaApplication implements CommandLineRunner {
15     @Autowired
16     private CategoryRepository categoryRepository;
17
18     @Autowired
19     private ArticleRepository articleRepository;
20
21     public static void main(String[] args) {
22         SpringApplication.run(SpringShopJpaApplication.class, args);
23     }
24
25     @Override
26     public void run(String... args) throws Exception {
27         categoryRepository.save(new Category("Smartphone"));
28         articleRepository.save(new Article("S9", "Samsung", 250));
29     }
30 }
```

```
MySQL Client (MariaDB 10.3 (x64)) - mysql -u root -p
2 rows in set (0.002 sec)

MariaDB [shop]> select * from article;
Empty set (0.047 sec)

MariaDB [shop]> select * from article;
+----+-----+-----+-----+
| id | brand | description | price |
+----+-----+-----+-----+
| 1  | Samsung | S9          | 250   |
+----+-----+-----+-----+
1 row in set (0.000 sec)

MariaDB [shop]> select * from category;
+----+-----+
| id | name   |
+----+-----+
| 1  | Smartphone |
+----+-----+
1 row in set (0.000 sec)

MariaDB [shop]>
```

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> SpringShopJpaApplication [Java Application] C:\Users\moham\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin\javaw.exe (29 juil. 2021,
2021-07-29 15:57:55.079 INFO 6404 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2021-07-29 15:57:55.350 INFO 6404 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-07-29 15:57:55.477 INFO 6404 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-07-29 15:57:55.504 INFO 6404 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MariaDB103Dialect
2021-07-29 15:57:55.905 INFO 6404 --- [main] org.hibernate.tuple.PojoInstantiator : HHH000182: No default (no-argument) constructor for class: fr.ldnr.entities.Article
2021-07-29 15:57:56.122 INFO 6404 --- [main] org.hibernate.tuple.PojoInstantiator : HHH000182: No default (no-argument) constructor for class: fr.ldnr.entities.Category
2021-07-29 15:57:56.363 INFO 6404 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.jta.platform.internal.NoJtaPlatform]
2021-07-29 15:57:56.375 INFO 6404 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-07-29 15:57:56.930 INFO 6404 --- [main] fr.ldnr.SpringShopJpaApplication : Started SpringShopJpaApplication in 4.529 seconds (JVM
Hibernate: insert into category (name) values (?)
Hibernate: insert into article (brand, description, price) values (?, ?, ?)
2021-07-29 15:57:57.168 INFO 6404 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
```

NB : EntityManagerFactory modélise l'unité de persistance. C'est à partir de lui que l'on peut construire des objets de type EntityManager, permettant d'interagir avec la base.

STEP 7 : SPRING DATA, CONCRÈTEMENT COMMENT ÇA MARCHE ?

On souhaite par ex ajouter une méthode qui renvoie l'ensemble des articles de telle marque, la requête sql n'est pas complexe et cela devrait être similaire avec Spring data :



```
public interface ArticleRepository extends JpaRepository<Article, Long> {  
    public List<Article> findByBrand(String brand);  
}
```

En effet, à partir de la méthode « findByBrand » Spring data comprend qu'il y a un select(find) et retourne la même chose qu'ici :

*select * from article where brand='Samsung';*

Ce qui se traduit par un simple appel dans le programme principale :

```
27 @Override  
28 public void run(String... args) throws Exception {  
29     //categoryRepository.save(new Category("Smartphone"));  
30     for(Article article : articleRepository.findByBrand("Samsung")) {  
31         System.out.println(article);  
32     }  
33 }  
34 }
```

Console Problems Debug Shell

<terminated> SpringShopJpaApplication [Java Application] C:\Users\moham\p2\pool\plugins\org.eclipse.j

```
2021-07-29 17:09:05.630 INFO 10744 --- [main] .s.d.r.c.RepositoryConf:  
2021-07-29 17:09:06.241 INFO 10744 --- [main] o.hibernate.jpa.interna:  
2021-07-29 17:09:06.317 INFO 10744 --- [main] org.hibernate.Version  
2021-07-29 17:09:06.516 INFO 10744 --- [main] o.hibernate.annotations.  
2021-07-29 17:09:06.745 INFO 10744 --- [main] com.zaxxer.hikari.Hikar:  
2021-07-29 17:09:06.844 INFO 10744 --- [main] com.zaxxer.hikari.Hikar:  
2021-07-29 17:09:06.866 INFO 10744 --- [main] org.hibernate.dialect.D:  
2021-07-29 17:09:07.463 INFO 10744 --- [main] org.hibernate.tuple.Poj:  
2021-07-29 17:09:07.622 INFO 10744 --- [main] o.h.e.t.j.p.i.JtaPlatfor  
2021-07-29 17:09:07.633 INFO 10744 --- [main] j.LocalContainerEntityMa  
2021-07-29 17:09:08.317 INFO 10744 --- [main] fr.ldnr.SpringShopJpaApp  
Hibernate: select article0_.id as id1_0_, article0_.brand as brand2_0_, article0_.  
Article [id=1, description=S9, brand=Samsung, price=250.0]  
Article [id=2, description=S8, brand=Samsung, price=250.0]
```

```
MariaDB [shop]> select * from article where brand='Samsung';  
+----+-----+-----+-----+  
| id | brand | description | price |  
+----+-----+-----+-----+  
| 1  | Samsung | S9          | 250   |  
| 2  | Samsung | S8          | 250   |  
+----+-----+-----+-----+  
2 rows in set (0.000 sec)
```

NB : Spring data analyse la méthode et utilise la requête HQL adéquate pour récupérer les données

STEP 7 : SPRING DATA

[Voir la doc ici](#)

```
MariaDB [shop]> select * from article where brand like '%sung%';
```

id	brand	description	price
1	Samsung	S9	250
2	Samsung	S8	250

2 rows in set (0.000 sec)



```
MariaDB [shop]> select * from article where brand='Samsung';
```

id	brand	description	price
1	Samsung	S9	250
2	Samsung	S8	250
4	Samsung	GalaxyTab	350

3 rows in set (0.000 sec)

```
public interface ArticleRepository extends JpaRepository<Article,Long> {  
    public List<Article> findByBrand(String brand);  
    public List<Article> findByBrandContains(String brand);  
}
```

```
MariaDB [shop]> select * from article where brand='Samsung' and price=250;
```

id	brand	description	price
1	Samsung	S9	250
2	Samsung	S8	250

```
public interface ArticleRepository extends JpaRepository<Article,Long> {  
    public List<Article> findByBrand(String brand);  
    public List<Article> findByBrandContains(String brand);  
    public List<Article> findByBrandAndPrice(String brand, double price);  
}
```

```
for(Article article : articleRepository.findByBrandAndPrice("Samsung",250)) {  
    System.out.println(article);  
}
```

```
Article [id=1, description=S9, brand=Samsung, price=250.0]  
Article [id=2, description=S8, brand=Samsung, price=250.0]
```

Quel est le résultat ici →

```
for(Article article : articleRepository.findByBrandAndPriceGreaterThan("Samsung",300)) {  
    System.out.println(article);  
}
```

STEP 7 : SPRING DATA / CAS PARTICULIER

Comment gérer les cas particuliers ?

En effet, les requêtes peuvent être parfois complexe aussi nous pouvons les coder nous même en Hql ([Hibernate Query Language](#))

Nous recherchons tous les articles dont la marque contient un mot clé(ici « sung » et dont le prix est supérieur à une valeur donnée (ici 200) :

Requête Sql

```
select * from Article where brand like '%sung%' and price>200;
```

Requête Hql

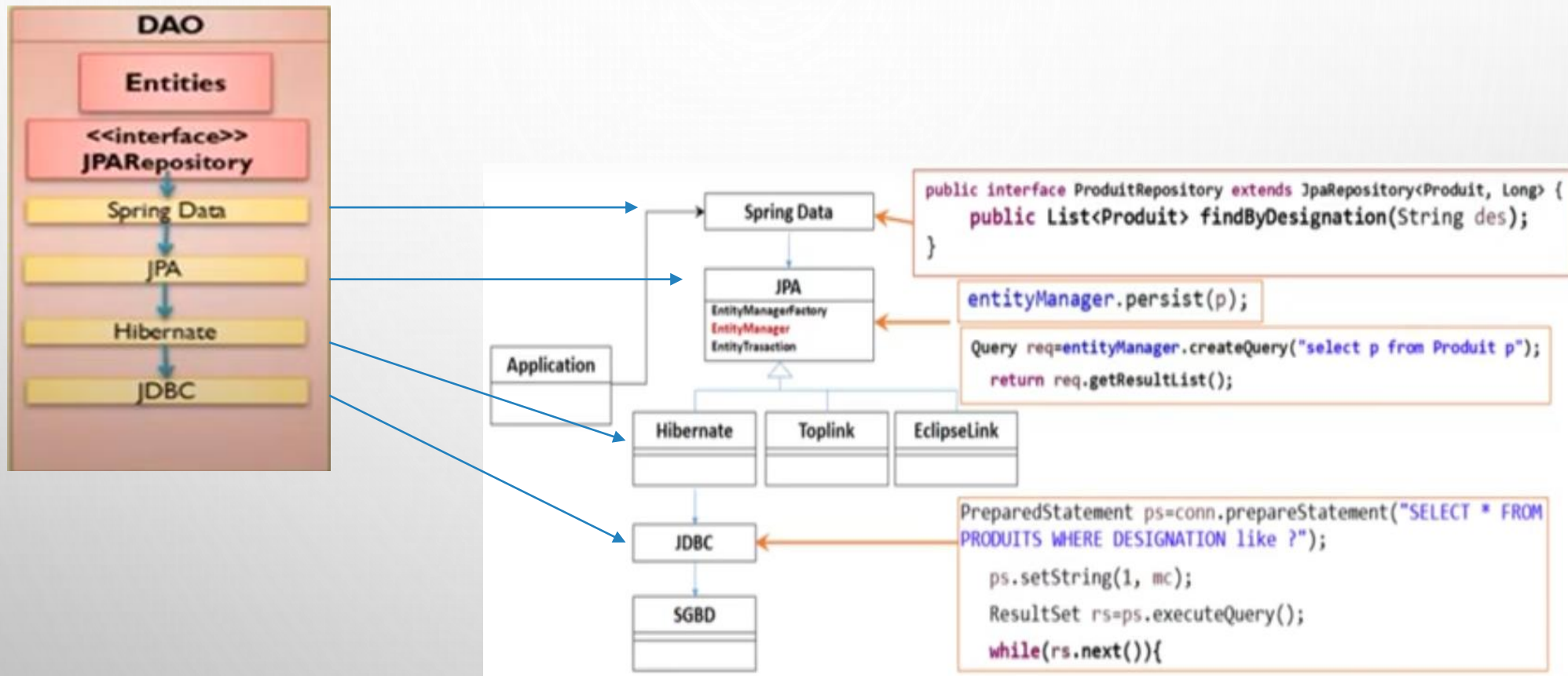
```
@Query("select A from Article A where A.brand like %:x% and A.price > :y")  
public List<Article> searchArticles(@Param("x") String kw, @Param("y")double price);
```

```
for(Article article : articleRepository.searchArticles("sung",200)) {  
    System.out.println(article);  
}
```

Rappelons tout de même qu'une bonne partie des cas peuvent régler avec « findBy.. »

D'après vous, quelle est la méthode correspondante ?

En résumé, spring data est une interface et une implémentation générique permettant de gérer des entités JPA.
Automatisant du coup, toute une partie du code !



C'est l'illustration de l'intérêt d'un framework

STEP 8 : MISE EN ŒUVRE DES ANNOTATIONS

@OneToMany & @ManyToOne

pour répondre à ce besoin :

« Ajouter des catégories d'articles puis afficher tous les articles d'une catégorie »

Jpa a crée une colonne de jointure, category_id est la clé étrangère et correspond à la clé primaire de category

```
MariaDB [shop]> select * from article;
+----+-----+-----+-----+-----+
| id | brand | description | price | category_id |
+----+-----+-----+-----+-----+
| 1  | Samsung | S11         | 750   | 8            |
| 2  | Samsung | S9          | 350   | 1            |
| 3  | Xiaomi  | MI10        | 100   | 1            |
| 4  | Samsung | GalaxyTab   | 450   | 2            |
| 5  | Apple   | Ipad        | 350   | 2            |
| 6  | Asus    | R510        | 600   | 3            |
| 7  | Samsung | S9          | 150   | 6            |
+----+-----+-----+-----+-----+
7 rows in set (0.101 sec)

MariaDB [shop]> select * from category;
+----+-----+
| id | name      |
+----+-----+
| 1  | Smartphone |
| 2  | Tablet    |
| 3  | PC        |
| 6  | Smartphone |
| 7  | Smartphone |
+----+-----+
```

```
@Entity
public class Category {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    @OneToMany(mappedBy = "category")
    private Collection<Article> articles; //une categorie est liée à plusieurs articles

    public Category() {
    }

    public Category(String name) {
        this.name = name;
    }
}
```

Esclave

Association bidirectionnelle 1 : N

Complément « mapping objet relationnel »

```
Article.java
@Entity
public class Article implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;
    private String brand;
    private double price;

    @ManyToOne
    private Category category; //plusieurs articles sont liés à une seule categorie

    public Article(String description, String brand, double price, Category category) {
    }
}
```

Maitre

NB : Les relations entre entités, telles que définies en JPA peuvent être unidirectionnelles ou bidirectionnelles. Dans ce second cas, l'une des deux entités doit être esclave(1) et l'autre maître (N) L'entité esclave doit préciser l'attribut mappedBy qui référence le champ qui porte la relation côté maître.

Ajoutons un jeu d'essai et vérifions les tables en base

```
@Override
public void run(String... args) throws Exception {
    Category smartphone = categoryRepository.save(new Category("Smartphone"));
    Category tablet = categoryRepository.save(new Category("Tablet"));
    Category pc = categoryRepository.save(new Category("PC"));

    articleRepository.save(new Article("S10", "Samsung", 500, smartphone));
    articleRepository.save(new Article("S9", "Samsung", 350, smartphone));
    articleRepository.save(new Article("MI10", "Xiaomi", 100, smartphone));

    articleRepository.save(new Article("GalaxyTab", "Samsung", 450, tablet));
    articleRepository.save(new Article("Ipad", "Apple", 350, tablet));

    articleRepository.save(new Article("R510", "Asus", 600, pc));
}
```

```
MariaDB [shop]> select * from article;
+----+-----+-----+-----+-----+
| id | brand | description | price | category_id |
+----+-----+-----+-----+-----+
| 1  | Samsung | S10      | 500   | 1           |
| 2  | Samsung | S9       | 350   | 1           |
| 3  | Xiaomi  | MI10     | 100   | 1           |
| 4  | Samsung | GalaxyTab | 450   | 2           |
| 5  | Apple   | Ipad     | 350   | 2           |
| 6  | Asus    | R510     | 600   | 3           |
+----+-----+-----+-----+-----+
6 rows in set (0.000 sec)

MariaDB [shop]> select * from category;
+----+-----+
| id | name  |
+----+-----+
| 1  | Smartphone |
| 2  | Tablet  |
| 3  | PC      |
+----+-----+
```

Reste à mettre en œuvre un « findBy » sur mesure pour avoir tous les articles d'une catégorie

```
public List<Article> findByCategoryId(Long categoryId);
```

```
Article [id=1, description=S10, brand=Samsung, price=500.0]
Article [id=2, description=S9, brand=Samsung, price=350.0]
Article [id=3, description=MI10, brand=Xiaomi, price=100.0]
```

RESSOURCES

- LIVRE « SPRING-PAR-LA-PRATIQUE-ED2 »
- [HTTPS://WWW.JMDOUDOUX.FR/JAVA/DEJ/INDEX.HTM](https://www.jmdoudoux.fr/java/dej/index.htm)
- [HTTPS://FR.SLIDESHARE.NET/MOHAMEDYOUSSEFI9/SUPPORT-DE-COURS-SPRING-MYOUSSEFI](https://fr.slideshare.net/mohamedyoussefi9/support-de-cours-spring-myoussfi)
- [HTTPS://WWW.YOUTUBE.COM/CHANNEL/UCCWIYNPQVHZTD3VX_KRNMDA](https://www.youtube.com/channel/UCCWIYNPQVHZTD3VX_KRNMDA)