Support Formation JAVA Spring Hibernate

Pour	Formation
Date	10/10/2024
Objet	Support Formation JAVA Spring Hibernate

I) Glossaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.

- **ORM**: Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.

- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le en HQL le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.

- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.

- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. En effet selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.

- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.

II) Intérêt de Spring avec un ORM JPA

Une des grandes applications du Framework Spring est le fait de gérer les transactions via une source de données.

A quoi sert le Framework Spring combinées avec l'ORM JPA Hibernate ?

1) Construire et injecter un **EntityManager** à travers un fichier de configuration que l'on peut nommer en général par **applicationContext.xml**.
2) Gérer les transactions de l'**EntityManager**.
3) Gérer la fermeture de ressources telles que l'**EntityManagerFactory** et l'**EntityManager**.

Considérons le programme suivant :

```
EntityManager em = null;
EntityManagerFactory emf = null;
try {
	emf = Persistence.createEntityManagerFactory("MonUniteDePersistance");
	em = emf.createEntityManager();

	em.getTransaction().begin();

	operationNumero1();
	operationNumero2();
	operationNumero3();
	operationNumero4();
	operationNumero5();
	operationNumero6();

	em.getTransaction().commit();

} catch (Exception e) {
	 if(em !=null){
		em.getTransaction().rollback();
	}
} finally {
	if(emf !=null){
		emf.close();
	}
	if(em !=null){
		em.close();
	}
}
```

Avec le Framework Spring nous aurons :

```
SpringCreeMonEntityManager();
SpringDebuteUneTransaction();

operationNumero1();
operationNumero2();
operationNumero3();
operationNumero4();
operationNumero5();
operationNumero6();

SpringCommitLaTransactionOuLeRollBack();
SpringFermeLesRessources();
```

Il est donc possible de s'affranchir de la gestion des transactions avec **Spring**.
Nous allons dans un premier temps télécharger les librairies nécessaires pour l'utilisation du Framework **Spring/JPA**.

III) Application : Mise en place d'un DAO Spring/Hibernate

Récupérer le projet **maven-dao-personnes-jpa-hibernate** que vous aviez implémenté dans le cours **Support-Formation-Java-JPA-Hibernate.pdf**. Si vous avez fini l'implémentation du projet **maven-personnes-dao-jpa-hibernate** c'est excellent nous allons pouvoir avancer sinon il faut terminer l'implémentation de **maven-dao-personnes-jpa-hibernate** avant de faire la suite.

Créer le projet Maven **maven-dao-personnes-jpa-spring-hibernate.**

1) Mise à jour du fichier pom.xml.

On ajoute la liste des Jar à telecharger pour utiliser **Spring/JPA**, le « **pom.xml** » de **maven-gestion-personnes-dao-jpa-spring** devient donc :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cours.spring</groupId>
  <artifactId>maven-personnes-dao-spring-jpa-hibernate</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
    <spring.version>4.1.4.RELEASE</spring.version>
    <hibernate.version>4.0.1.Final</hibernate.version>
    <mysql.version>5.1.41</mysql.version>
  </properties>
  <dependencies>
    <!-- Debut Spring dependencies -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-orm</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <!-- Fin Spring dependencies -->
    <!-- Debut Hibernate dependencies -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>${hibernate.version}</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
```

```
        <version>${hibernate.version}</version>
      </dependency>
      <!-- Fin Hibernate dependencies -->
      <!-- Debut MYSQL dependencies -->
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>${mysql.version}</version>
      </dependency>
      <!-- Debut MYSQL dependencies -->
      <!-- Debut logger log4j dependencies -->
      <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
      </dependency>
      <dependency>
        <groupId>log4j</groupId>
        <artifactId>apache-log4j-extras</artifactId>
        <version>1.2.17</version>
      </dependency>
      <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.2</version>
      </dependency>
      <!-- Fin logger log4j dependencies -->
    </dependencies>
</project>
```

2) Copie de **maven-personnes-dao-jpa-hibernate/src** vers **maven-personnes-dao-jpa-spring-jpa-hibernate/src**.

Le contenu du fichier **persistence.xml** sera maintenant :

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="PersonnesPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>com.cours.dao.entities.Personne</class>
  </persistence-unit>
</persistence>
```

Ligne 2 : la balise racine du fichier XML est **<persistence>**.
Ligne 3 : **<persistence-unit>** sert à définir une unité de persistance. Il peut y avoir plusieurs unités de persistance. Chacune d'elles a un nom (attribut name) et un type de transactions (attribut transaction-type). L'application aura accès à l'unité de persistance via le nom de celle-ci, ici JPA. Le type de transaction **RESOURCE_LOCAL** indique que l'application gère elle-même les transactions avec le SGBD. Ce sera le cas ici. Lorsque l'application s'exécute dans un conteneur EJB3, elle peut utiliser le service de transactions de celui-ci. Dans ce cas, on mettra transaction-type=**JTA** (Java Transaction API).
JTA est la valeur par défaut lorsque l'attribut transaction-type est absent.

Ligne 4 : la balise **<provider>** sert à définir une classe implémentant l'interface [javax.persistence.spi.PersistenceProvider], interface qui permet à l'application d'initialiser la couche de persistance. Parce qu'on utilise une implémentation JPA / Hibernate, la classe utilisée ici est une classe d'Hibernate.

Ligne 5 : la balise **<properties>** introduit des propriétés propres au provider particulier choisi. Ainsi selon qu'on a choisi **Hibernate, Toplink, EclipseLink, Kodo**, ... on aura des propriétés différentes. Celles qui suivent sont propres à Hibernate.

Pour éviter d'eventuelles problèmes de configuration nous allons renommer le fichier « **persistence.xml** » en « **spring-persistence.xml** ».
Ceci va nous permetter d'éviter d'éventuelles conflits entre le conteneur Spring et le conteneur du serveur d'application(serveur d'application avec un conteneur à Servlet et un conteneur EJB, exemple : GlassFish, JBoss ect…) dans une application Web Http Servlet. Nous n'aurons pas besoins de fermer les ressources liés à la base de données car Spring s'en chargera et c'est ce principe de Spring qu'on appelle l'inversion de contrôle (IOC). C'est plutôt magique non !

3) Modification des classes **PersonneDao** et **ServiceFacade**.

Ajouter à la classe DAO **PersonneDao** les annotation **@Repository("personneDao")**, **@Transactional** et **@PersistenceContext**. Il faudra aussi mettre à jour le code de cette classe pour éliminer la gestion des transaction et la fermeture de l'**EntityManager**.

L'annotation **@Transactional** ce qui veut dire que toutes methodes de la classe PersonneDao sont transactionnelles.
L'annotation **@ PersistenceContext** au-dessus de l'attribut « **javax.persistence.EntityManager em** » veut dire qu'un objet [EntityManager] est injecté dans le champ **em** grâce à cette annotation JPA. La classe DAO **PersonneDao** est instanciée une unique fois. C'est un singleton utilisé par tous les threads utilisant la couche JPA. Ainsi donc l'EntityManager **em** est commun à tous les threads. Sping gére aussi cette aspect de Multithreading qui est quand meme un des problèmes majeur en programmation Java.

Ajouter à la classe Service **ServiceFacade** les annotation **@Service("serviceFacade")** et **@Autowired**. Il faudra aussi mettre à jour le code de cette classe pour utiliser **Spring** pour injecter le Bean de type **IPersonneDao**.

Les annotations **Component**, **Repository**, **Service** sont des annotations Spring. Elles permettent à Spring de détecter les classes à instancier.

4) Suppression du package **com.cours.dao.factory**.

Supprimer le package **com.cours.dao.factory**. En effet l'utilisation d'un DAO Factory n'est plus necessaire car désormais ce sera à Spring de jouer ce rôle.

5) Création du fichier **maven-personnes-dao-jpa-spring-jpa-hibernate/src/main/resources/applicationContext.xml**.

Le fichier **applicationContext.xml** peut avoir comme contenu :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <context:component-scan base-package="com.cours.spring.service" />

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="persistenceXmlLocation" value="classpath*:META-INF/spring-persistence.xml"/>
        <property name="dataSource" ref="dataSource" />
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                <!-- -->
                <property name="showSql" value="true" />
                <property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
                <property name="generateDdl" value="true" />
            </bean>
        </property>
        <property name="loadTimeWeaver">
            <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver" />
        </property>
    </bean>
    <!-- Simple implementation of the standard JDBC DataSource interface,
    configuring the plain old JDBC DriverManager via bean properties -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/base_personnes_jpa?useSSL=false" />
        <property name="username" value="application" />
        <property name="password" value="passw0rd" />
    </bean>
    <!-- le gestionnaire de transactions -->
    <tx:annotation-driven transaction-manager="txManager" />
    <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>
    <!-- traduction des exceptions -->
    <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
    <!-- persistence -->
    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
</beans>
```

En version copiable :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-4.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.2.xsd">

  <context:component-scan base-package="com.cours.spring.service" />

  <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceXmlLocation" value="classpath*:META-INF/spring-persistence.xml"/>
    <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter">
      <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <!-- -->
        <property name="showSql" value="true" />
        <property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
        <property name="generateDdl" value="true" />
      </bean>
    </property>
    <property name="loadTimeWeaver">
      <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver" />
    </property>
  </bean>
  <!-- Simple implementation of the standard JDBC DataSource interface,
  configuring the plain old JDBC DriverManager via bean properties -->
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/base_personnes_jpa?useSSL=false" />
    <property name="username" value="application" />
    <property name="password" value="passw0rd" />
  </bean>
  <!-- le gestionnaire de transactions -->
  <tx:annotation-driven transaction-manager="txManager" />
  <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
  </bean>
  <!-- traduction des exceptions -->
  <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
  <!-- persistence -->
  <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
</beans>
```

Ligne 13 : Definition des repertoires que Spring devra scanner pour rechercher d'éventuelles annotations ici ce sera **com com.cours.spring.service**.
Ligne 15 : Création de « entityManagerFactory » de la librairie **JPA** ici Hibernate.
Ligne 32-37 : Configuration de la source de données qui correspond ici à une base de données MYSQL.
Ligne 40-42 : Configuration des transactions qui sont maintenant géré par **Spring**. On voit bien l'injection du bean « entityManagerFactory » qui a été crée précédamment.

La classe de démarrage **MainDao** aura comme contenu :

```java
package com.cours.main;

import com.cours.dao.entities.Personne;
import com.cours.dao.IPersonneDao;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainDao {

    public static String className = MainDao.class.getName();

    public static void main(String[] args) {
        String methodName = "main";
        boolean isDeleteOk;
        ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
        IPersonneDao personneDao = (IPersonneDao) ctx.getBean("personneDao");
        Personne personneCRUD = new Personne("Barro", "Barro", 75, "rue des passeurs", "Laval", "53000");
        personneCRUD = personneDao.create(personneCRUD);
        personneCRUD.setPrenom("Barro Bis");
        personneCRUD.setNom("Barro Bis");
        personneCRUD = personneDao.update(personneCRUD);
        isDeleteOk = personneDao.delete(personneCRUD);
        System.out.println(className + " --> " + methodName + " , personnes : " + personneDao.findAll() + " , isDeleteOk : " + isDeleteOk);
    }
}
```

En version copiable :

```java
package com.cours.main;

import com.cours.dao.entities.Personne;
import com.cours.dao.IPersonneDao;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainDao {

   public static String className = MainDao.class.getName();

   public static void main(String[] args) {
      String methodName = "main";
      boolean isDeleteOk;
      ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
      IPersonneDao personneDao = (IPersonneDao) ctx.getBean("personneDao");
      Personne personneCRUD = new Personne("Barro", "Barro", 75, "rue des passeurs", "Laval", "53000");
      personneCRUD = personneDao.create(personneCRUD);
      personneCRUD.setPrenom("Barro Bis");
      personneCRUD.setNom("Barro Bis");
      personneCRUD = personneDao.update(personneCRUD);
      isDeleteOk = personneDao.delete(personneCRUD);
      System.out.println(className + " --> " + methodName + " , personnes : " + personneDao.findAll() + " ,
isDeleteOk : " + isDeleteOk);
   }
}
```

On obtient à l'éxécution du Main :

```
------------------------------------------------------------------------
Building maven-gestion-personnes-dao-spring 1.0-SNAPSHOT
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ maven-gestion-personnes-dao-spring ---
mai 13, 2019 9:27:55 PM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFOS: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@5910e440: startup date [Mon May 13 21:27:5
mai 13, 2019 9:27:55 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFOS: Loading XML bean definitions from class path resource [applicationContext.xml]
mai 13, 2019 9:27:57 PM org.springframework.jdbc.datasource.DriverManagerDataSource setDriverClassName
INFOS: Loaded JDBC driver: com.mysql.jdbc.Driver
mai 13, 2019 9:27:57 PM org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean createNativeEntityManagerFactory
INFOS: Building JPA container EntityManagerFactory for persistence unit 'PersonnesPU'
mai 13, 2019 9:27:57 PM org.hibernate.annotations.common.Version <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {4.0.1.Final}
mai 13, 2019 9:27:57 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {4.0.1.Final}
mai 13, 2019 9:27:57 PM org.hibernate.cfg.Environment <clinit>
INFO: HHH000206: hibernate.properties not found
mai 13, 2019 9:27:57 PM org.hibernate.cfg.Environment buildBytecodeProvider
INFO: HHH000021: Bytecode provider name : javassist
mai 13, 2019 9:27:57 PM org.hibernate.ejb.Ejb3Configuration configure
INFO: HHH000204: Processing PersistenceUnitInfo [
        name: PersonnesPU
        ...]
mai 13, 2019 9:27:57 PM org.hibernate.service.jdbc.connections.internal.ConnectionProviderInitiator instantiateExplicitConne
INFO: HHH000130: Instantiating explicit connection provider: org.hibernate.ejb.connection.InjectedDataSourceConnectionProvic
mai 13, 2019 9:27:57 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
mai 13, 2019 9:27:57 PM org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService
INFO: HHH000268: Transaction strategy: org.hibernate.engine.transaction.internal.jdbc.JdbcTransactionFactory
mai 13, 2019 9:27:57 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
INFO: HHH000397: Using ASTQueryTranslatorFactory
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
INFO: HHH000228: Running hbm2ddl schema update
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
INFO: HHH000102: Fetching database metadata
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
INFO: HHH000396: Updating schema
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.TableMetadata <init>
INFO: HHH000261: Table found: base_personnes_jpa.personne
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.TableMetadata <init>
INFO: HHH000037: Columns: [ville, idpersonne, taille, rue, codepostal, poids, prenom, nom, version, pays]
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.TableMetadata <init>
INFO: HHH000108: Foreign keys: []
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.TableMetadata <init>
INFO: HHH000126: Indexes: [primary]
mai 13, 2019 9:27:58 PM org.hibernate.tool.hbm2ddl.SchemaUpdate execute
INFO: HHH000232: Schema update complete
com.cours.spring.main.MainDaoSpring --> main , personnes : [[idPersonne=1, prenom=Julien, nom=Marshall, poids=55.0, rue=rue
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 4.544s
```