

Atelier Framework Spring-03

JPA –Partie03

Objectifs

- **Implémenter le DAO (Approche 1)**
 - Créer une interface DAO
 - Créer une implémentation DAO

1. Créer, dans « **src/main/java** », un package «**spring.jpa.dao**»
2. Définir l'interface « **IProduitDao** », dans le package «**spring.jpa.dao**», pour déclarer les traitements à réaliser sur l'entité « **Produit** » (**déclarer le QUOI**):

```
package spring.jpa.dao;

import java.util.List;

import spring.jpa.model.Produit;

public interface IProduitDao
{
    Produit save (Produit p);
    List<Produit> findAll();
    Produit findOne(Long id);
    Produit update (Produit p);
    void delete (Long id);
    List<Produit> findByDesignation( String mc);
}
```

3. Créer, dans le même package, une classe nommée «**ProduitDaoImpl**» qui implémente l'interface «**IProduitDao**» et présente les caractéristiques suivantes :
 - Elle est annotée par « **@Repository** » pour indiquer à **Spring** qu'il s'agit d'un **bean** à mettre dans le contexte et à être instancié par **Spring** via l'injection de dépendance.

- Elle est annotée par « **@Transactional** » pour indiquer à Spring que toutes les méthodes sont transactionnelles (pour gérer les « **commit** » et les « **rollback** »).
- Elle déclare un objet « **em** » de type « **EntityManager** » qui reçoit une instance par injection de dépendance dynamique à travers l'annotation « **@PersistenceContext** ».
- L'interface « **EntityManager** » possède les méthodes adéquates pour envoyer des requêtes SQL (persist, merge,...).

4. Voici le code de la classe d'implémentation :

```
package spring.jpa.dao;
import java.util.List;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.Query;
import jakarta.transaction.Transactional;
import org.springframework.stereotype.Repository;
import spring.jpa.model.Produit;

@Repository
@Transactional
public class ProduitDaoImpl implements IProduitDao{

    @PersistenceContext
    private EntityManager em;
    public Produit save(Produit p) {
        em.persist(p);
        return p;
    }
    public List<Produit> findAll() {
        Query query=
        em.createQuery("select p from Produit p order by p.designation");
        return query.getResultList();
    }
    public Produit findOne(Long id) {
        Produit p = em.find(Produit.class, id);
        return p;
    }
    public Produit update(Produit p) {
        em.merge(p);
        return p;
    }
    public void delete(Long id) {
        Produit p = em.find(Produit.class, id);
        em.remove(p);
    }
}
```

```

    public List<Produit> findByDesignation(String mc) {
        Query query=
em.createQuery("select p from Produit p where p.designation like :x");
        query.setParameter("x", "%"+mc+"%");
        return query.getResultList();
    }
}

```

5. Passer maintenant à l'exécution de l'application. Pour se faire, modifier le code de la classe «**JpaSpringBootApplication**» comme suit (pour insérer des produits et réaliser des traitements en utilisant le composant DAO):

```

package spring.jpa;

import java.util.List;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import spring.jpa.dao.IProduitDao;
import spring.jpa.model.Produit;

@SpringBootApplication
public class JpaSpringBootApplication {

    public static void main(String[] args) {
        // référencer le contexte
        ApplicationContext contexte=
        SpringApplication.run(JpaSpringBootApplication.class, args);
        //Récupérer une implémentation de l'interface "IProduitDao" par injection de dépendance
        IProduitDao daoProduit = contexte.getBean(IProduitDao.class);
        // Insérer 3 produits
        daoProduit.save(new Produit("Yahourt", 0.400, 20));
        daoProduit.save(new Produit("Farine", 1.200, 30));
        daoProduit.save(new Produit("Chocolat", 2000.0, 5));
        // Lister l'ensemble des produits
        System.out.println("***Liste de tous les produits:");
        System.out.println("*****");
        List<Produit> lp = daoProduit.findAll();
        for (Produit p : lp)
        {
            System.out.print("Designation:"+ p.getDesignation()+" , ");
            System.out.println("Prix:"+ p.getPrix());
        }
        // Lister tous les produits dont la designation contient "h"
        System.out.println("-----");
        System.out.println("***Liste de tous les produits dont la designation contient 'h':");
        System.out.println("*****");

        List<Produit> lp2 = daoProduit.findByDesignation("h");
        for (Produit p : lp2)
        {
            System.out.print("Designation:"+ p.getDesignation()+" , ");
            System.out.println("Prix:"+ p.getPrix());
        }
    }
}

```

6. Afin de réduire les messages de journalisation dans la console (les log »), créer un fichier «**logBack.xml**» sous «**src/main/resources** » ayant le simple code suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>

</Configuration>
```

7. Pour cacher les messages SQL dans la console, il suffit de mettre la propriété « **spring.jpa.show-sql** » à la valeur « **false** » dans le fichier « **application.properties** ».
8. Pour cacher le logo de Spring dans la console, ajouter, dans le fichier « **application.properties** », la ligne suivante :

spring.main.banner-mode=off

Exercice

9. Ajouter dans l'interface, «**IProduitDao**», la déclaration de la méthode :

```
List<Produit> findByDesignationAndPrix( String mc, double prix);
```

Cette méthode retourne tous les produits ayant une désignation contenant le mot «**mc**» et ayant un prix supérieur à «**prix**».

10. Implémenter cette méthode puis passer au test de son fonctionnement