# JAVA 4A SAGI – Application en couches / Maven / Spring

*Ce document présente les éléments suivants*
*- architecture en couches*
*- gestion des couches avec Spring*

## 1) Principe du découplage Metier / DAO

**couche DAO :** donne accés aux données
(souvent dans des bases de données)

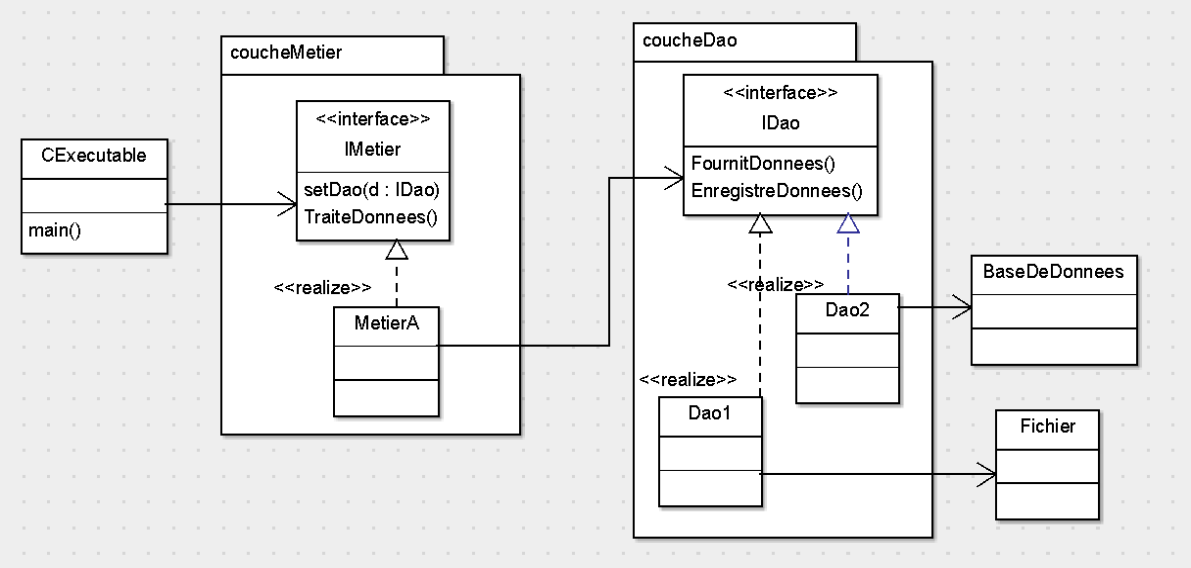
**couche Metier** (ou de traitement): lien sur la couche DAO et contient les règles métier

**Découplage des couches :** les couches doivent pouvoir être remplacées facilement. Pour assurer ce *découplage*, chaque couche est décrite par une ***interface*** que la couche doit implémenter. Les accès à la couche se font via une référence de "type interface". Le type (dynamique) réel importe peu.

Chacune des couches de l'application peut être remplacée sans que les autres soient impactées.

**IDao :** interface de la couche d'accès aux données (DAO)
**IMetier :** interface de la couche métier

### *Diagramme de principe*




**La fonction `main()` dialogue uniquement avec la couche Metier (via une référence `IMetier`).**

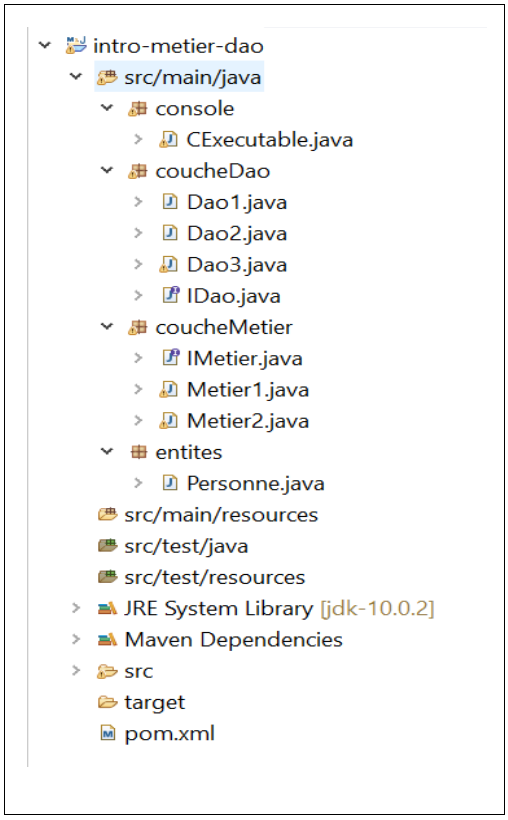
```
package console;
import coucheDao.*;
import coucheMetier.*;
public class CExecutable {
	public static void main(String[] args)
	{	// Création des couches + lien Metier->Dao
		IDao dao = new Dao1();
		IMetier metier = new MetierA();
		metier.setDao(dao);

		// le programme utilise ensuite seulement la référence metier
		metier.TraiteDonnees();
		...
	}
}
```

**Important:** chaque couche est remplaçable par une autre couche qui implémente la même interface.

## 2) Exemple à deux couches : Metier / DAO

Créer un projet Maven et *y a*jouter les types fournis.

**groupID = `pta.sagi`**
artifactID = **`intro-metier-dao`**




**Couche DAO** : accès à des objets **`Personne(nom,age)`**.

**Couche Métier :** Peut lire/écrire les données DAO et faire un traitement sur les données

**Remarque**: la couche Metier a ses propres données, éventuellement différentes de celles de Dao.

**`IDao`** : **`getAll`** (lecture des données)
**`setAll`** (écriture données)
**`getByAge`** (lecture avec filtre)

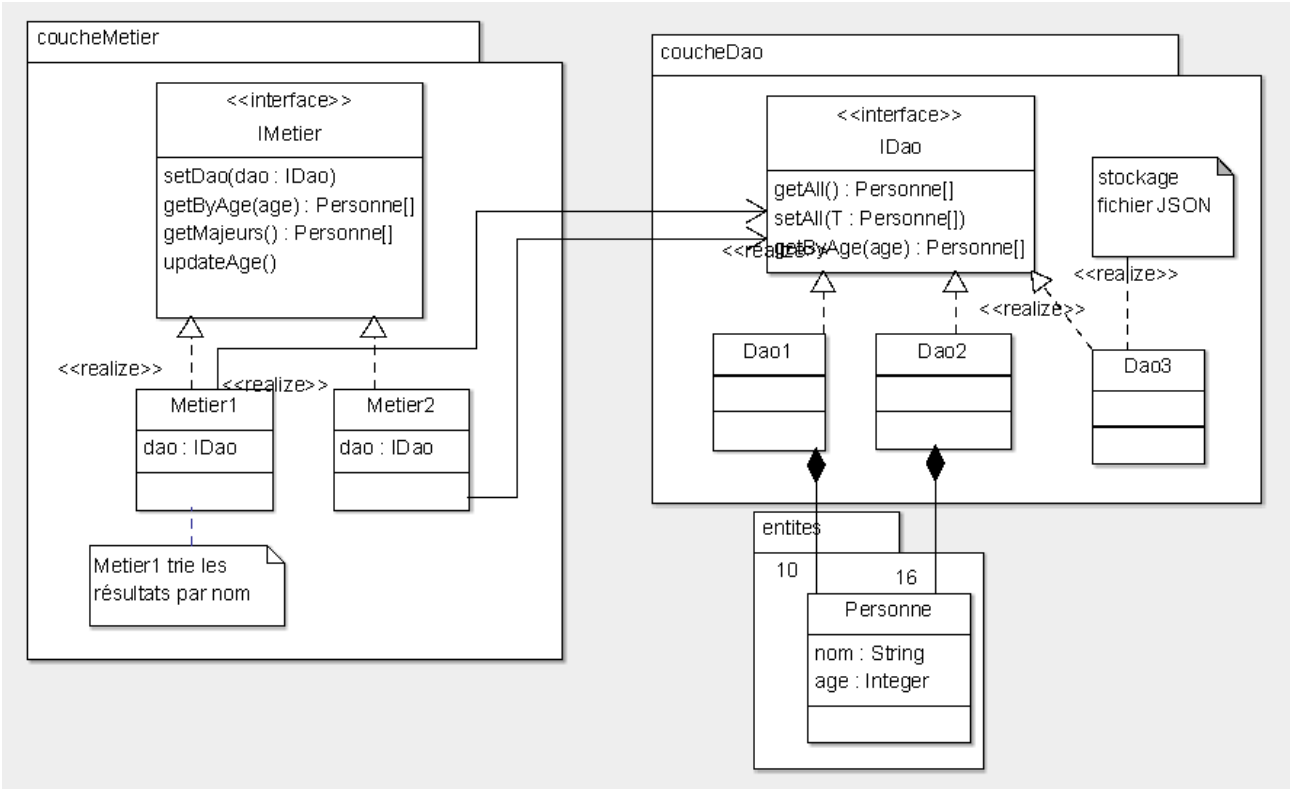
**`IMetier`** : **`getByAge`** (redirection vers DAO)
**`getMajeurs`** (lit et filtre données)
**`updateAge`** (lit puis écrit des données)

**3 implémentations de la couche Dao** (seule Dao3 est persistante)**:**

**`Dao1`**: 10 objets Personne
**`Dao2`** : 16 objets Personne d'âge tiré aléatoirement
**`Dao3`** : objets Personne lus/écrits dans fichier JSON

**2 implémentations de la couche Métier :**

**`Metier1`** : les résultats **sont triés** par nom
**`Metier2`** : les résultats sont dans l'ordre fourni par DAO



**Ex:** interfaces des couches et implémentations Dao2 et Metier2

```java
//IDao.java  [interface couche Dao]
package coucheDao;

import entites.Personne;

public interface IDao {
 public Personne[] getAll();
 public void setAll(Personne[] personnes);
 public Personne[] getByAge(int age);
}
```

```java
//IMetier.java  [interface couche Metier]
package coucheMetier;
import coucheDao.IDao;
import entites.Personne;

public interfaceIMetier {
 public void setDao(IDao dao);
 public Personne[] getByAge(int age);
 public Personne[] getMajeurs();
 public void updateAge();
}
```

```java
//Dao2.java
package coucheDao;
import entites.Personne;
public class Dao2 implements IDao {

 private Personne[] personnes;

 public Dao2()
 {
 personnes=new Personne[18];
 String[] prenoms = new String[]{"Marie","Pierre",
                        "Lucie", "Louis","Gustave","Léon"};

 Random rand = new Random();
 for(int i=0;i<18;i++){
  personnes[i]= new Personne(prenoms[rand.nextInt(6)],
                                        rand.nextInt(50)+2);
  }
 }

 public Personne[] getAll() {
        return personnes.clone();
 }

 public void setAll(Personne[] personnes){
   if(personnes!=null)
        this.personnes = personnes.clone();
 }

 public Personne[] getByAge(int age) {
   ArrayList<Personne> pers=new ArrayList<Personne>();
   for(Personne p:this.personnes){
     if(p.getAge()==age)pers.add(p);
   }
  return pers.toArray(new Personne[pers.size()]);
 }
}
```

```java
//Metier2.java
package coucheMetier;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

import coucheDao.IDao;
import entites.Personne;

public class Metier2 implements IMetier
{
  private IDao dao;

  public void setDao(IDao dao){
        this.dao =dao;
  }

  public Metier2() {}

  public Personne[] getByAge(int age){
        return dao.getByAge(age);
  }

  public Personne[] getMajeurs() {
    Personne[] personnes=dao.getAll();
    ArrayList<Personne> tab;
    tab=new ArrayList<Personne>();
    for(Personne p:personnes){
        if(p.getAge()>=18) tab.add(p);
    }
    return tab.toArray(new Personne[0]);
  }

  public void updateAge() {
    Personne[] personnes=dao.getAll();
    for(Personne p:personnes)
         p.setAge(p.getAge()+1);
    dao.setAll(personnes);
   }
}
```

## Création des couches et utilisation de la couche métier (main)

```
//CExecutable.java [classe "executable" = celle de main()]

package console;
import coucheMetier.*;
import coucheDao.*;
import entites.*;
import java.io.*;

public class CExecutable
{
        public static void PrintData(Personne[] tabP){
                System.out.println("Données:");
                for(Personne p:tabP) System.out.println(p.toString());
        }

        public static void main(String[] args)  throws IOException
        {
        // CREATION COUCHES de l'APPLICATION
        IDao dao= new Dao2();
        IMetier metier = new Metier2();
        metier.setDao(dao);

        int choix;
        String prompt="0:Quit 1:Majeurs 2:UpdateAge";
        String line;
        BufferedReader clavier =
                new BufferedReader(new InputStreamReader(System.in));

        do
        {
                System.out.println(prompt);
                try{
                  line=clavier.readLine().trim();
                  choix= Integer.parseInt(line);
                }
                catch(Exception e){     choix=0; }

                switch(choix)
                {
                case 1:
                        PrintData(metier.getMajeurs());
                        break;
                case 2:
                        metier.updateAge();
                        break;
                default:
                        choix=0;
                        break;
                }
        }while(choix!=0);

        System.out.println("Bye!");
}
```

***Sortie Console***

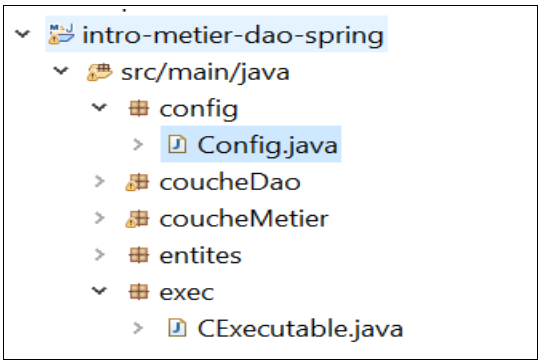
```
0:Quit 1:Majeurs 2:UpdateAge
1
Données:
nom=Pierre age=34
nom=Lucie age=32
nom=Léon age=27
nom=Gustave age=38
nom=Gustave age=19
nom=Louis age=48
nom=Marie age=45
nom=Gustave age=23
nom=Louis age=29
nom=Gustave age=29
nom=Marie age=40
0:Quit 1:Majeurs 2:UpdateAge
2
0:Quit 1:Majeurs 2:UpdateAge
1
Données:
nom=Pierre age=35
nom=Lucie age=33
nom=Pierre age=18
nom=Léon age=28
nom=Gustave age=39
nom=Gustave age=20
nom=Louis age=49
nom=Marie age=46
nom=Gustave age=24
nom=Louis age=30
nom=Gustave age=30
nom=Marie age=41
0:Quit 1:Majeurs 2:UpdateAge
0
Bye!
```

## 3) Projet Maven/Spring (version 1)

**Framework Spring** : crée et lie les objets des différentes couches
**Rôle de Maven :** gérer les dépendances sur Spring.

### Créer un projet Maven

groupID = **pta.sagi**
artifactID=**intro-metier-dao-spring**




### Ajouter les classes Metier/Dao

### Ajouter les classes suivantes [code page suivante]

**config.Config** [config. Spring]
**exec.CExecutable** [classe exécutable]

### Le fichier POM.xml (Maven) est donné ci-dessous

```
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>pta.sagi</groupId>
  <artifactId>intro-metier-dao-spring</artifactId>
  <version>0.1</version>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>2.3.12.RELEASE</version>
  </parent>

   <dependencies>
   <dependency>
              <groupId>org.springframework</groupId>
              <artifactId>spring-context</artifactId>
   </dependency>

   <dependency>
              <groupId>com.fasterxml.jackson.core</groupId>
              <artifactId>jackson-databind</artifactId>
   </dependency>
</dependencies>
</project>
```

**Classe de configuration Spring : indique quels objets instancier**

```
package config;    //Config.java [classe de config. Spring]

import coucheMetier.*;
import coucheDao.*;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration            //Annotation Spring
public class Config {

       @Bean IDao dao(){
              return new Dao2();}

       @Bean IMetier metier(IDao dao){
              IMetier m=new Metier2();
              m.setDao(dao);
              return m;
       }
}
```

***Un bean Spring = un objet instancié par Spring***

**Classe exécutable :** on récupère le contexte Spring (paramétré par la classe de configuration)

Ce **contexte Spring** donne accès aux **beans**, c'est-à-dire aux objets instanciés par Spring.

```
package exec;

import java.util.Arrays;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import config.Config;
import coucheMetier.IMetier;
import entites.Personne;

public class CExecutable {

       public static void main(String[] args) {
        IMetier metier=null;
        AnnotationConfigApplicationContext ctx;
        ctx = new AnnotationConfigApplicationContext(Config.class);
        try{
              metier = ctx.getBean(IMetier.class);

              // Programme console
              ...
        }
        catch(Exception ex){
              System.out.println(ex.getMessage());
              System.exit(11);
        }
        ctx.close();
       }

}
```

## 4) Projet Maven/Spring (version 2) Lien par injection de dépendance.

L'objet de la couche métier doit avoir une référence sur l'objet de la couche DAO. On peut utiliser ***l'injection de dépendance*** Spring pour faire ce lien de façon automatique. Il suffit d'annoter par **`@Autowired`** la référence concernée.

**Note** : la méthode `IMetier.setDao(dao)` devient alors inutile

```
package coucheMetier;
import coucheDao.IDao;
import entites.Personne;

public interface IMetier {
	//public void setDao(IDao dao);
	public Personne[] getByAge(int age);
	public Personne[] getMajeurs();
	public void updateAge();
}
```

```
package coucheMetier;

import org.springframework.beans.factory.annotation.Autowired;
import coucheDao.IDao;
import entites.Personne;

public class Metier2 implements IMetier
{
	@Autowired	// injection du bean IDao
	private IDao dao;

	/* inutile
	public void setDao(IDao dao) {
		this.dao =dao;
	} */
	...
}
```

```
package config;    //[classe de config. Spring]
import coucheMetier.*;
import coucheDao.*;
import ...

@Configuration      //Annotation Spring
public class Config {

  @Bean IDao dao(){return new Dao2();}

  @Bean IMetier metier(){return new Metier2();}
}
```

**Note :** la classe de configuration est plus simple, puisque le lien entre les couches est fait par @Autowired (injection de dépendance)

**Classe exécutable :** pas de différence avec la version 1.

```
package lancement;

import ...

public class CExecutable {

  public static void main(String[] args) {
	IMetier metier=null;
	AnnotationConfigApplicationContext ctx=null;
	try
	{	ctx = new AnnotationConfigApplicationContext(Config.class);
		metier =ctx.getBean(IMetier.class);

		//Programme console
		...

	}
	catch(Exception ex)
	{	System.out.println(ex.getMessage());
		System.exit(11);
	}
	finally{ if(ctx!=null) ctx.close();}
}
```

## 5) Projet Maven/Spring (version 3) : annoter les classes des beans par @Component.

Spring crée un bean de chaque classe annotée **@Component**. Il n'y a plus explicitement à demander à créer des beans particuliers dans la classe de configuration. En revanche, on doit indiquer dans quels packages chercher ces annotations @Component.

Note : il ne doit donc y avoir qu'une seule classe ainsi annotée par couche.

```
package coucheMetier;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import coucheDao.IDao;
import entites.Personne;

@Component
public class Metier1 implements IMetier {

        @Autowired
        private IDao dao;

        ...
```

```
package coucheDao;

import entites.Personne;
import org.springframework.stereotype.Component;

@Component
public class Dao1 implements IDao
{
   ...
}
```

```
package config;    //[ classe de configuration Spring]

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;


@Configuration
@ComponentScan({"coucheDao","coucheMetier"})
public class Config
{
  // les beans "@Component" sont cherchés dans les packages
  // indiqués à l'annotation @ComponentScan
}
```

```
package lancement;    //CExecutable.java  [INCHANGEE]
...
public class CExecutable {

public static void main(String[] args) {
  IMetier metier=null;
  AnnotationConfigApplicationContext ctx=null;
  try{
    ctx = new AnnotationConfigApplicationContext(Config.class);
    metier =ctx.getBean(IMetier.class);

       ...
```

## 6) [Extra] Ajout couche DAO (stockage en base MySQL)

On peut ajouter une quatrième couche DAO (nouvelle implémentation de IDao) afin de stocker les données des personnes dans une base de données MySQL.

On propose de compléter le projet précédent en y ajoutant la nouvelle couche Dao.

### Ce qui est ajouté : Dao4 (implémentation MySQL)

#### 1) Créer la base de données des personnes

Exécuter le script fourni (copié ci-contre)

```
-- Création de base
CREATE DATABASE IF NOT EXISTS db_personne
DEFAULT CHARACTER SET utf8 COLLATE utf8_swedish_ci;
--
USE db_personne;

-- Création de table
CREATE TABLE IF NOT EXISTS personne
(
 id INTEGER NOT NULL AUTO_INCREMENT,
 nom VARCHAR(15) NOT NULL DEFAULT '?',
 age SMALLINT NOT NULL,
 PRIMARY KEY (id)
)
ENGINE=InnoDB  DEFAULT CHARSET=utf8
COLLATE=utf8_swedish_ci AUTO_INCREMENT=1;

-- PEUPLER la table
INSERT INTO personne (nom,age) VALUES
('Bécassine',45),('Milou',7),('Jacques',28),
('Pierre',34),('Romain',27),('Louis',15),
('Marie',32),('Elodie',38),('Alexandre',25);
```

#### 2) Ajouter la dépendance

```
mysql / mysql-connector-java
```

#### 3) Modifier la classe de configuration Spring

```
@Configuration
public class Config {

	@Bean
	IDao dao(){ return new Dao4();}   //le bean DAO

	@Bean
	IMetier metier(){ return new Metier1();}

}
```

#### 4) Ajouter la classe Dao4 qui implémente IDao

En copie sur les pages suivantes.

```
// Dao4
package coucheDao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import entites.Personne;

public class Dao4 implements IDao
{
	final static String url="jdbc:mysql://localhost:3306/db_personne";
	final static String user="root";
	final static String pwd="root";

	public Dao4(){
		try{ Class.forName("com.mysql.jdbc.Driver");}
		catch(ClassNotFoundException ex){
			System.out.println(ex.getMessage());
			System.exit(1); // driver non trouvé
		}
	}
	public Personne[] getAll() {
		String select = "SELECT NOM,AGE FROM personne";
		ArrayList<Personne> alp = new ArrayList<Personne>();
		// try-with-resources
		try(Connection con=DriverManager.getConnection(url,user,pwd);
			PreparedStatement ps = con.prepareStatement(select);
			ResultSet rs=ps.executeQuery();)
		{

			while(rs.next()){
					Personne p=new Personne(rs.getString(1),rs.getInt(2));
					alp.add(p);
				}
				return alp.toArray(new Personne[0]);
		}
		catch(SQLException e1){
				System.out.println(e1.getMessage());
				return null;
		}
	}

	public void setAll(Personne[] personnes) {
		String drop = "DELETE FROM personne";
		String insert = "INSERT INTO personne (id,nom,age) VALUES (?,?,?)";
		ArrayList<Personne> alp = new ArrayList<Personne>();
		// try-with-resources
		try(Connection con=DriverManager.getConnection(url,user,pwd))
		{
			con.setAutoCommit(false);

			try(PreparedStatement ps = con.prepareStatement(drop);)
			{
				ps.executeUpdate();
			}
			catch(SQLException e1){
				con.rollback();
				System.out.println(e1.getMessage());
			}
```

```
                        try(PreparedStatement ps = con.prepareStatement(insert);)
                        {
                          int id=1;
                          for(Personne p:personnes)
                          {
                                ps.setInt(1, id);
                                ps.setString(2,p.getNom());
                                ps.setInt(3, p.getAge());
                                ps.executeUpdate();
                                id++;
                          }
                        }
                        catch(SQLException e1){
                                con.rollback();
                                System.out.println(e1.getMessage());
                        }
                        con.commit();
                        con.setAutoCommit(true);
                }
                catch(SQLException e1)
                {

                        System.out.println(e1.getMessage());
                }
        }

        @Override
        public Personne[] getByAge(int age) {
                String select = "SELECT NOM,AGE FROM personne where age=?";
                ArrayList<Personne> alp = new ArrayList<Personne>();
                // try-with-resources
                try(Connection con=DriverManager.getConnection(url,user,pwd);
                        PreparedStatement ps = con.prepareStatement(select);)
                {
                        ps.setInt(1,age); //définit l'age

                        try(ResultSet rs=ps.executeQuery();){
                                while(rs.next()){
                                        Personne p=new Personne(rs.getString(1),rs.getInt(2));
                                        alp.add(p);
                                }
                                return alp.toArray(new Personne[0]);
                        }
                        catch(SQLException e1)
                        {
                        System.out.println(e1.getMessage());
                        return new Personne[0];
                        }
                }
                catch(SQLException e1){
                                System.out.println(e1.getMessage());
                                return new Personne[0];
                }
        }

}
```

## 7) [Extra] GUI JavaFX avec les couches Metier/DAO

L'objectif est de créer une interface utilisateur avec JavaFX pour communiquer avec les couches Metier/DAO. L'utilisateur peut cliquer sur deux boutons "Majeurs" et "Update", les résultats sont sortis dans un contrôle ListView

1) créer projet Maven pta.sagi/jfx-metier-dao-spring/0.1

2) Ajouter une dépendance à pta.sagi/intro-metier-dao-spring/0.1

3) Ajouter un package application

application.Control = classe de controller JavaFx
application.Main = classe exécutable

4) Fichier ui.fxml (Scene Builder) dans le dossier du package application.

Controller class = application.Control
control ListView (fx:id=listview)
control Button "Majeurs" (On Action = OnMajeurs)
control Button "Update" (On Action = Update)

5) Créer une classe de configuration Spring (package config)

config.ConfigJfx : définit les beans Dao et Metier

6) La classe "controller"

```
package application;
...
public class Control
{
	@Autowired
	IMetier metier;   // référence sur le bean Metier

	@FXML
	ListView<Personne> listview;

	@FXML
	void OnMajeurs(ActionEvent e){
		Personne[] tab=metier.getMajeurs();
		listview.getItems().clear();
		for(Personne p:tab)  listview.getItems().add(p);
	}

	@FXML
	void OnUpdate(ActionEvent e){
		metier.updateAge();
		listview.getItems().clear();
	}
}
```

7) la classe exécutable application.Main

Note: à ce niveau, on peut qualifier de "bricolage" la façon de lier l'instance application.Control (le contrôleur) au bean Metier créé par Spring.

```
package application;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
...


public class Main extends Application {

	private AnnotationConfigApplicationContext context;

	@Override
	public void stop(){  context.close(); }

	@Override
	public void start(Stage primaryStage) {
	try {
		context = new AnnotationConfigApplicationContext(ConfigJfx.class);

		FXMLLoader loader = new FXMLLoader(getClass().getResource("ui.fxml"));
		Pane root = (Pane)loader.load();
		Scene scene = new Scene(root,500,300);
		primaryStage.setScene(scene);
		primaryStage.setTitle("JavaFx Spring");

		Control top = loader.getController();
		context.getAutowireCapableBeanFactory().autowireBean(top);

		primaryStage.show();
	} catch(Exception e) {
		e.printStackTrace();
	}
	}

	public static void main(String[] args){
		Application.launch(args);
	}
}
```