# TP 2 : Implémentation de l'Inversion de Contrôle et Injection des Dépendances avec Spring

Pr. Mohamed LACHGAR

## Objectif

L'objectif est de créer une application Java avec Spring, mettant en œuvre l'inversion de contrôle (IoC) et l'injection de dépendances (DI) via des annotations. Il s'agira de développer une interface DAO, une interface métier, et de gérer l'injection des dépendances entre elles.

## Étapes du TP

### Pré-requis

Avant de commencer, il est nécessaire d'installer :

- **Eclipse IDE for Java Developers**
- **JDK 11** ou une version supérieure
- **Maven** pour la gestion des dépendances

### Création du projet dans Eclipse

1. Ouvrir Eclipse et créer un nouveau projet Maven. Sélectionner l'archetype `quickstart` pour obtenir un projet de base.
2. Nommer le projet par exemple `SpringDIProject`. Définir le `Group Id` (ex. `com.example`) et l'`Artifact Id` (ex. `springdi`).
3. Configurer le fichier `pom.xml` en y ajoutant les dépendances nécessaires à Spring.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.22</version>
    </dependency>
</dependencies>
```

Listing 1: Dépendances dans `pom.xml`

Mettre à jour les dépendances avec `Maven → Update Project`.

## Création des interfaces et classes Java

### 1. Interface `IDao`

Créer un package nommé `dao`, puis y ajouter l'interface `IDao`.

```java
package dao;

public interface IDao {
    double getValue();
}
```

Listing 2: Interface `IDao`

### 2. Classe `DaoImpl`

Ajouter une classe `DaoImpl` dans le même package, qui implémente `IDao`, et utiliser l'annotation `@Component` pour en faire un bean Spring.

```java
package dao;

import org.springframework.stereotype.Component;

@Component("dao")
public class DaoImpl implements IDao {
    @Override
    public double getValue() {
        return 100.0;
    }
}
```

Listing 3: Classe `DaoImpl`

### 3. Classe `DaoImpl2`

Ajouter une seconde implémentation `DaoImpl2` de l'interface `IDao`.

```
package dao;

import org.springframework.stereotype.Component;

@Component("dao2")
public class DaoImpl2 implements IDao {
    @Override
    public double getValue() {
        return 150.0;
    }
}
```

Listing 4: Classe `DaoImpl2`

### 4. Interface `IMetier`

Créer un package `metier` et y ajouter l'interface `IMetier`.

```
package metier;

public interface IMetier {
    double calcul();
}
```

Listing 5: Interface `IMetier`

### 5. Classe `MetierImpl`

Ajouter la classe `MetierImpl` qui implémente `IMetier`. Utiliser l'annotation `@Autowired` pour injecter l'instance de `IDao`, et `@Component` pour déclarer la classe comme bean.

```
package metier;

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

@Component("metier")
public class MetierImpl implements IMetier {

    @Autowired
    private IDao dao;
```

```
12
13          @Override
14          public double calcul() {
15              return dao.getValue() * 2;
16          }
17      }
```

Listing 6: Classe `MetierImpl`

## Configuration Spring avec annotations

### Classe principale `Presentation2`

Dans un package **`presentation`**, créer la classe principale **`Presentation2`**. Cette classe récupérera les beans Spring via un contexte de type **`AnnotationConfigApplicationContext`**.

```
1   package presentation;
2
3   import metier.IMetier;
4   import org.springframework.context.ApplicationContext;
5   import org.springframework.context.annotation.
        AnnotationConfigApplicationContext;
6   import org.springframework.context.annotation.
        ComponentScan;
7   import org.springframework.context.annotation.
        Configuration;
8
9   @Configuration
10  @ComponentScan(basePackages = {"dao", "metier"})
11  public class Presentation2 {
12      public static void main(String[] args) {
13          ApplicationContext context = new
                AnnotationConfigApplicationContext(
                Presentation2.class);
14          IMetier metier = context.getBean(IMetier.class);
15          System.out.println("Resultat = " + metier.calcul
                ());
16      }
17  }
```

Listing 7: Classe `Presentation2`

## Exécution du projet

Compiler et exécuter la classe **`Presentation2`**. Si le projet est correctement configuré, le résultat attendu dans la console sera :

```
Résultat = 200.0
```

# Bonus

- Ajouter des tests unitaires avec JUnit pour valider les implémentations de `DaoImpl` et `MetierImpl`.
- Créer une nouvelle implémentation de `IDao` et configurer Spring pour injecter dynamiquement cette nouvelle classe dans `MetierImpl`.
- Explorer l'utilisation de fichiers de configuration XML pour l'injection des dépendances.
- Implémenter un profil Spring pour activer/désactiver certaines implémentations selon l'environnement (développement, production, etc.).