

DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Filière Ingénieur :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Compte rendu du TP1



Inversion de contrôle et Injection des dépendances



Réalisé par :

• MOUAKKAL Nouhayla

Encadré par :

• Pr YOUSSEFI Mohamed

Année Universitaire : 2023 - 2024

Sommaire:

Introduction :	6
Partie 1 : Mise en œuvre de l'injection des dépendances	7
1. Créer l'interface IDao avec une méthode getDate	7
2. Créer une implémentation de cette interface	7
3. Créer l'interface IMetier avec une méthode calcul	8
4. Créer une implémentation de cette interface en utilisant le couplage faible	8
5. Faire l'injection des dépendances :	8
a. Par Instanciation Statique.....	9
b. Par Instanciation dynamique	9
c. En utilisant le Framework Spring	11
i. Version XML	11
ii. Version annotations	12
Conclusion	13

Introduction :

Ce compte rendu documente les différentes étapes de déploiement et de coL'inversion de contrôle (IoC) et l'injection de dépendances sont des concepts fondamentaux en développement logiciel, visant à améliorer la modularité, la flexibilité et la maintenabilité des applications. Ce compte rendu explore la mise en œuvre de l'injection de dépendances à travers un exemple concret, illustrant les différentes approches allant de l'instauration de l'interface IDao jusqu'à l'utilisation du framework Spring.

Dans la première partie, nous nous concentrerons sur la création de l'interface IDao avec une méthode getDate et son implémentation. De plus, nous explorerons la création de l'interface IMetier avec une méthode calcul et sa mise en œuvre en utilisant le couplage faible. L'étape cruciale de cette partie consiste à effectuer l'injection des dépendances. Nous aborderons cette démarche à travers trois méthodes distinctes : l'instanciation statique, l'instanciation dynamique, et enfin, l'utilisation du framework Spring, dans ses versions XML et annotations.

Ce compte rendu vise à offrir une compréhension approfondie des concepts IoC et d'injection de dépendances, tout en détaillant les différentes étapes de mise en œuvre, du design initial jusqu'à l'intégration de frameworks tels que Spring.

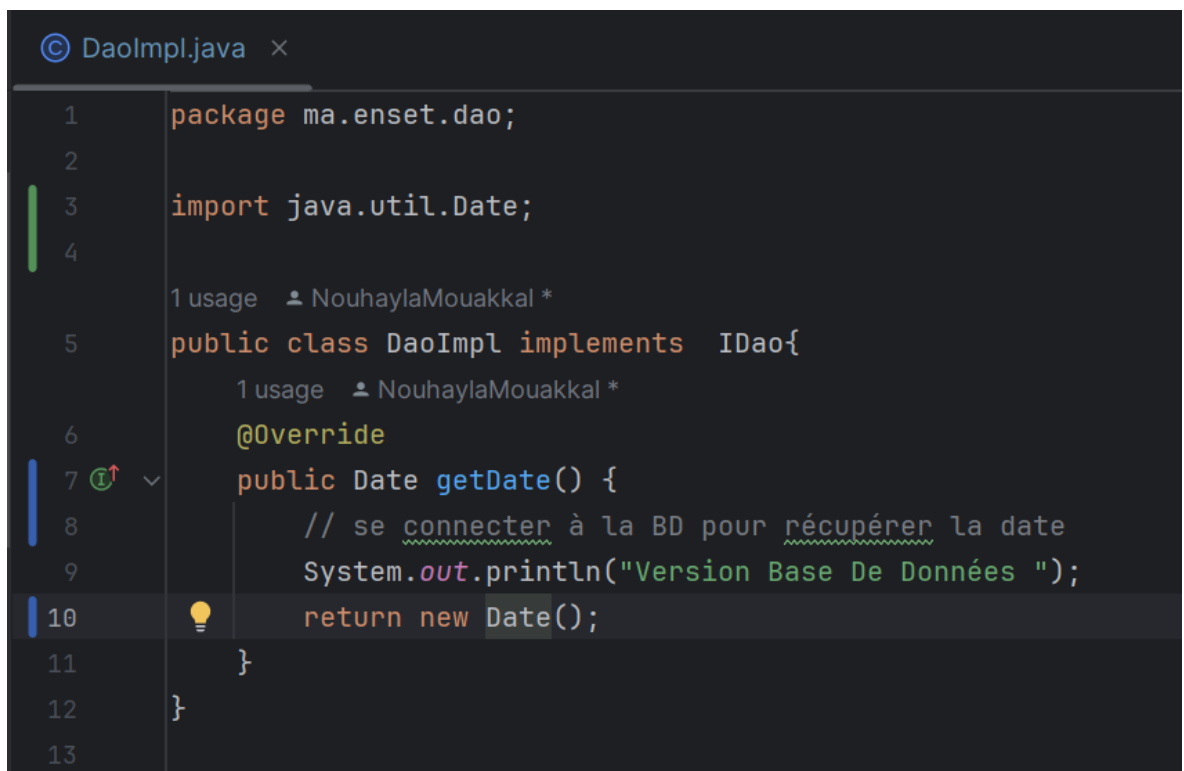
Partie 1 : Mise en œuvre de l'injection des dépendances

1. Créer l'interface IDao avec une méthode getDate



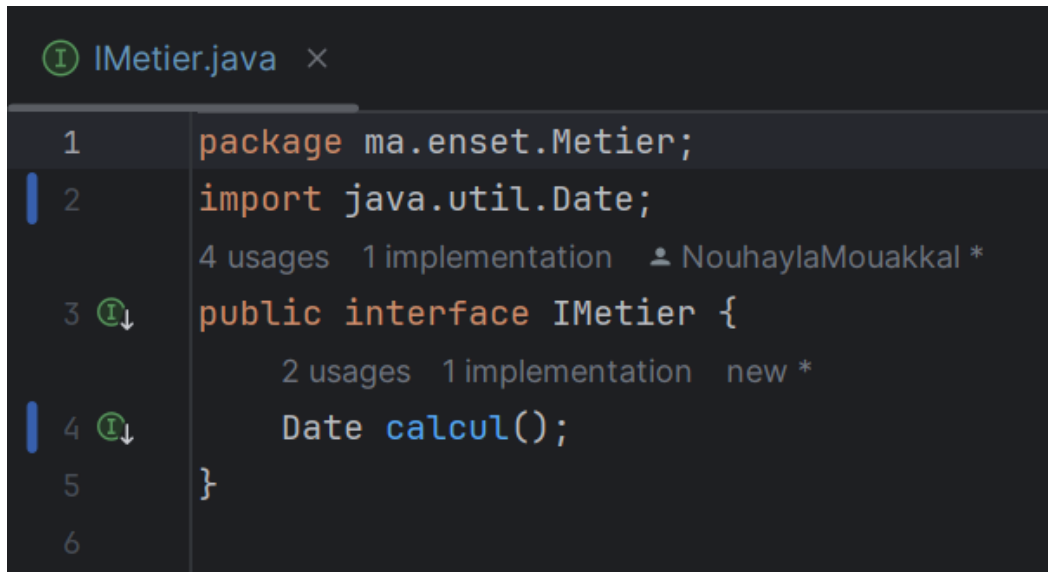
```
I IDao.java x
1 package ma.enset.dao;
2
3 import java.util.Date;
4
5 12 usages 3 implementations NouhaylaMouakkal *
6 public interface IDao {
7     1 usage 3 implementations new *
8     public Date getDate();
9 }
10
```

2. Créer une implémentation de cette interface



```
© DaoImpl.java x
1 package ma.enset.dao;
2
3 import java.util.Date;
4
5 1 usage NouhaylaMouakkal *
6 public class DaoImpl implements IDao{
7     1 usage NouhaylaMouakkal *
8     @Override
9     public Date getDate() {
10         // se connecter à la BD pour récupérer la date
11         System.out.println("Version Base De Données ");
12         return new Date();
13     }
14 }
15
```

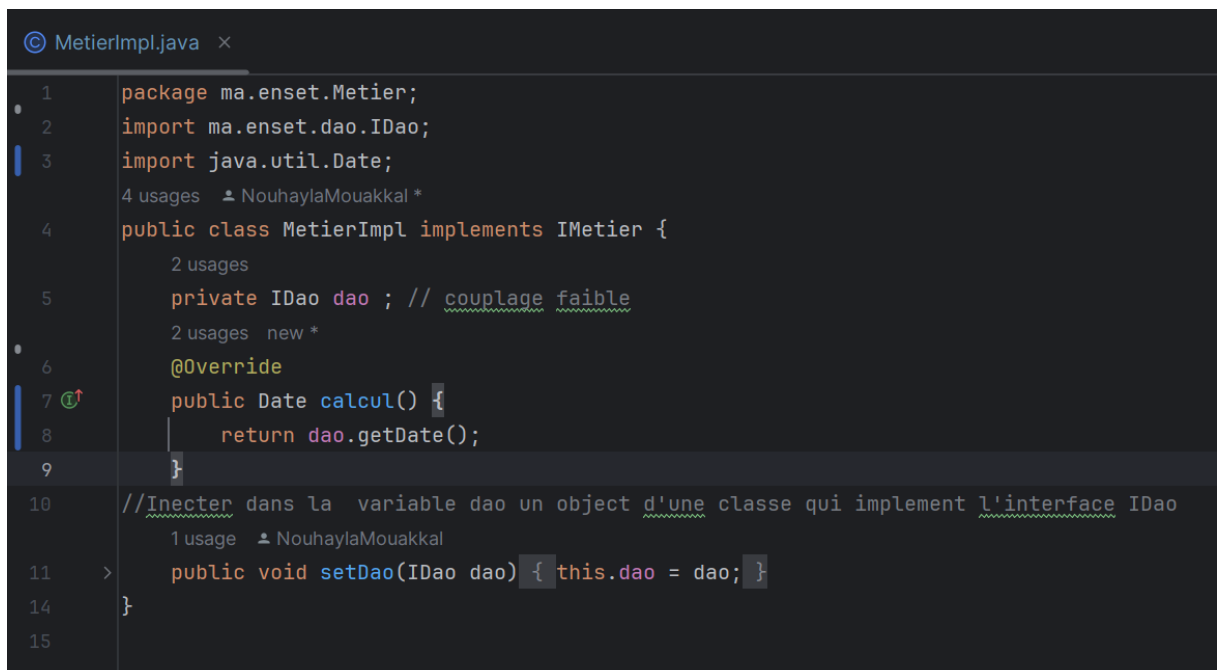
3. Créer l'interface IMetier avec une méthode calcul



```
I IMetier.java x
1 package ma.enset.Metier;
2 import java.util.Date;
3 public interface IMetier {
4     Date calcul();
5 }
6
```

The screenshot shows a code editor with a file named 'IMetier.java'. The code defines a package 'ma.enset.Metier', imports 'java.util.Date', and declares a public interface 'IMetier' with a single method 'calcul()' that returns a 'Date' object. The interface is enclosed in curly braces. The editor shows line numbers 1 through 6.

4. Créer une implémentation de cette interface en utilisant le couplage faible



```
MetierImpl.java x
1 package ma.enset.Metier;
2 import ma.enset.dao.IDao;
3 import java.util.Date;
4 public class MetierImpl implements IMetier {
5     private IDao dao ; // couplage faible
6     @Override
7     public Date calcul() {
8         return dao.getDate();
9     }
10    //Injecter dans la variable dao un object d'une classe qui implement l'interface IDao
11    public void setDao(IDao dao) { this.dao = dao; }
12
13
14
15
```

The screenshot shows a code editor with a file named 'MetierImpl.java'. The code defines a package 'ma.enset.Metier', imports 'ma.enset.dao.IDao' and 'java.util.Date', and declares a public class 'MetierImpl' that implements the 'IMetier' interface. The class has a private field 'dao' of type 'IDao' with a comment '// couplage faible'. It has an '@Override' annotation and a 'calcul()' method that returns 'dao.getDate()'. There is also a 'setDao(IDao dao)' method that sets 'this.dao = dao;'. The editor shows line numbers 1 through 15.

5. Faire l'injection des dépendances :

a. Par Instanciation Statique

```
© presentation.java x
1 package ma.enset.Presentation;
2 import ma.enset.Metier.MetierImpl;
3 import ma.enset.dao.DaoImpl;
4 import ma.enset.ext.DaoImpl2;
5
6 // NouhaylaMouakkal *
7 public class presentation {
8     // NouhaylaMouakkal *
9     public static void main(String[] args){
10        // Injection des dependances par instanciation statique => new => couplage fort
11        DaoImpl2 dao = new DaoImpl2();
12        MetierImpl metier = new MetierImpl();
13        metier.setDao(dao);
14        System.out.println("Résultat : "+metier.calcul());
15    }
16 }
```

Exécution

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\ja
Version Capteurs
Résultat : Mon Feb 19 11:26:05 WEST 2024

Process finished with exit code 0
```

b. Par Instanciation dynamique

D'abord on va créer un fichier config.txt qui contient le path des classes d'implémentation de l'interface

```
≡ config.txt x
1 ma.enset.dao.DaoImpl
2 ma.enset.Metier.MetierImpl
```

Ensuite on va créer une autre classe pour implémenter l'injection de dépendances dynamiquement

```
© pres2.java x
1 package ma.enset.Presentation;
2 import ma.enset.Metier.IMetier;
3 import ma.enset.dao.IDao;
4 import java.io.File;
5 import java.lang.reflect.Method;
6 import java.util.Scanner;
7
8 public class pres2 {
9     Scanner scanner = new Scanner(new File("src\\main\\java\\ma\\enset\\config.txt"));
10    String daoClassName= scanner.nextLine();
11    Class cDao = Class.forName(daoClassName);
12    IDao dao = (IDao) cDao.newInstance();
13    //System.out.println(dao.getDate());
14    String metierClassName = scanner.nextLine();
15    Class cMetier = Class.forName(metierClassName);
16    IMetier metier = (IMetier) cMetier.newInstance();
17
18    Method method = cMetier.getMethod("setDao", IDao.class);
19    method.invoke(metier,dao);
20    System.out.println("Résultat : "+ metier.calcul());
21 }
22 }
```

Exécution

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java
Version Base De Données
Résultat : Mon Feb 19 11:32:34 WEST 2024

Process finished with exit code 0
```

Si on veut la version Web Service on va juste modifier le fichier config.txt puis on exécute.

```
1 ma.enset.ext.DaoImplWS
2 ma.enset.Metier.MetierImpl
```

Exécution

```
"C:\Program Files\Java\jdk-18.0.1.1\bin\java
Version Web Service :
Résultat : Mon Feb 19 11:36:01 WEST 2024

Process finished with exit code 0
```

c. En utilisant le Framework Spring

Dans cette étape, on va initier un nouveau projet Maven et ajouter les dépendances requises de Spring, notamment spring.core, spring.bean, et spring.context, dans le fichier pom.xml.

```
1      <properties>
2          <maven.compiler.source>18</maven.compiler.source>
3          <maven.compiler.target>18</maven.compiler.target>
4          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
5      </properties>
6      <dependencies>
7          <dependency>
8              <groupId>org.springframework</groupId>
9              <artifactId>spring-core</artifactId>
10             <version>6.1.4</version>
11         </dependency>
12         <dependency>
13             <groupId>org.springframework</groupId>
14             <artifactId>spring-context</artifactId>
15             <version>6.1.4</version>
16         </dependency>
17         <dependency>
18             <groupId>org.springframework</groupId>
19             <artifactId>spring-beans</artifactId>
20             <version>6.1.4</version>
21         </dependency>
22     </dependencies>
```

i. Version XML

Pour cette version, il faut créer un fichier de configuration ApplicationContext.xml :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
5
6     <bean id="dao" class="ext.DaoImpl2"></bean>
7     <bean id="metier" class="Metier.MetierImpl">
8         <!-- <property name="dao" ref="dao"/> -->
9         <constructor-arg ref="dao"/></constructor-arg>
10    </bean>
11 </beans>
```

Puis on va créer une classe de présentation pour l'exécution contenant le code suivant :


```

PresentationSpringXML.java x
1 package Presentation;
2 import Metier.IMetier;
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 new *
7 public class PresentationSpringXML {
8     new *
9     public static void main(String[] args) {
10         ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "ApplicationContext.xml");
11         IMetier metier = (IMetier) context.getBean( name: "metier");
12         System.out.println("Result : " + metier.calcul());
13     }
14 }

```

Exécution

```

"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Version Capteurs
Result : Mon Feb 19 12:16:37 WEST 2024

Process finished with exit code 0

```

ii. Version annotations

Dans cette version il faut ajouter dans l'implémentation des interfaces l'annotation @Component

```

PresentationSpringAnnotations.java x
1 package Presentation;
2 import Metier.IMetier;
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 new *
7 public class PresentationSpringAnnotations {
8     new *
9     public static void main(String[] args) {
10         ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "dao", "metier");
11         IMetier metier = (IMetier) context.getBean(IMetier.class);
12         System.out.println("Result : " + metier.calcul());
13     }
14 }

```

Exécution

```

"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Version Base De Données
Result : Mon Feb 19 12:19:26 WEST 2024

Process finished with exit code 0

```

Conclusion

En conclusion, ce travail pratique nous a permis d'explorer de manière pratique les concepts fondamentaux d'inversion de contrôle et d'injection de dépendances. À travers la création d'interfaces, leur implémentation, et la manipulation des dépendances, nous avons pu expérimenter différentes approches, notamment le couplage faible et l'utilisation du framework Spring.

L'injection de dépendances se révèle être un outil puissant pour favoriser la modularité et la réutilisabilité du code. Les méthodes d'injection, qu'elles soient statiques, dynamiques, ou utilisant des frameworks comme Spring, offrent des solutions adaptées à divers contextes de développement.

Ce compte rendu souligne l'importance de comprendre ces concepts clés pour concevoir des applications évolutives et faciles à maintenir. L'intégration de l'inversion de contrôle et de l'injection de dépendances représente un pas significatif vers une conception logicielle plus robuste et flexible.