

Département Mathématiques et Informatiques

Compte Rendu de TP1

Architecture JEE et middleware

Filière d'ingénieur :

Ingénierie Informatique, Big Data et Cloud Computing

**Injection des dépendances,
Instanciation statique et dynamique**

Réalisé par : Abdellatif HASSANI

Encadré par : Mr. Mohamed YOUSSEFI

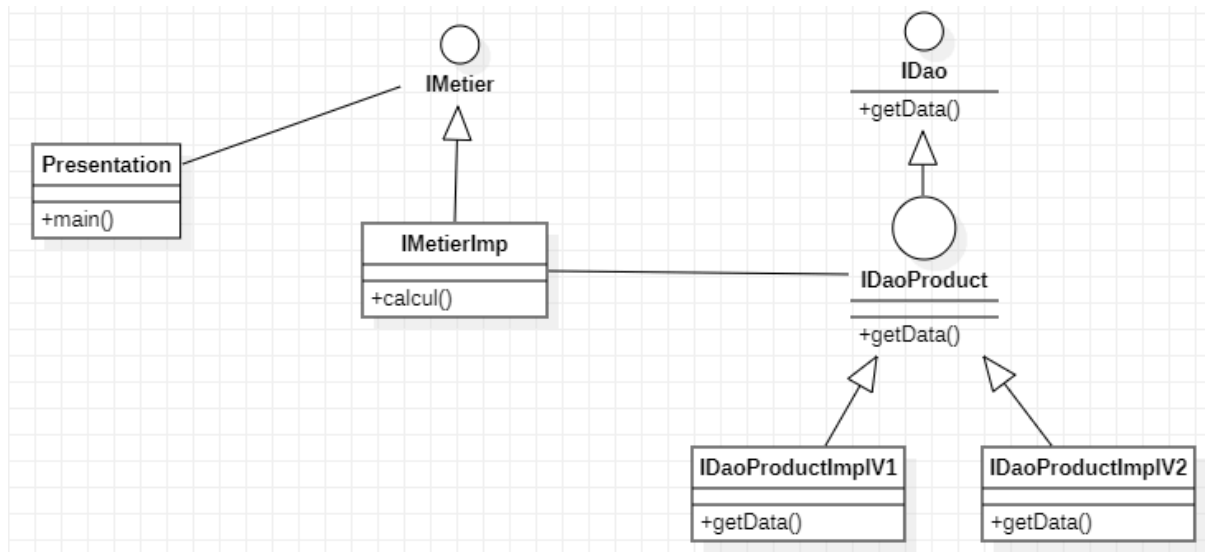
Année Universitaire : 2023-2024

Introduction

Le compte rendu du TP1 du module "Architecture JEE et Middleware" porte sur l'exploration des méthodes d'**injection de dépendances**, avec un focus sur les concepts d'**instanciation statique et dynamique**. L'objectif principal était de mettre en œuvre le principe de **couplage faible** en favorisant la dépendance aux interfaces plutôt qu'aux classes, afin de rendre le programme **fermé à la modification et ouvert à l'extension**. Cette approche permet aux classes dépendantes des interfaces de fonctionner avec n'importe quelle implémentation de l'interface. Le TP a débuté par l'instanciation statique, où les objets d'implémentation ont été créés et injectés via le constructeur ou le setter. Ensuite, nous avons exploré l'instanciation dynamique en utilisant un fichier de configuration pour spécifier les classes ou les implémentations à utiliser. En résumé, ce compte rendu met en avant notre compréhension et notre application pratique des concepts d'architecture JEE et de middleware, en insistant sur l'importance du couplage faible pour garantir la flexibilité et l'évolutivité des systèmes logiciels.

1) Conception

- Diagramme des classes



2) Code source

- Les entities

→ Product.java

```
package ma.enset.entities;

public class Product {

    private long id;

    private String name;

    private double price;

    private int quantity;

    public Product() {

    }

    //Getters and Setters and toString Method

}
```

- La couche dao

→ IDao.java

```
package ma.enset.dao;

import java.sql.SQLException;
import java.util.List;

public interface IDao<T, U> {

    List<T> getData() throws SQLException;

    void add(T o) throws SQLException;

}
```

→ IDaoProduct.java

```
package ma.enset.dao;

import ma.enset.entities.Product;

public interface IDaoProduct extends IDao<Product, Long> {

}
```

→ IDaoProductImplV1.java

```
package ma.enset.dao;

import ma.enset.entities.Product;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

//First Implementation of IDaoProduct interface in which data is
retrieved from the database

public class IDaoProductImplV1 implements IDaoProduct{

    //Retreiving data from DB

    @Override

    public List<Product> getData() throws SQLException {

        //Connecting to database
```

```

        Connection connection = SingletonDB.getConnection();

        //Creating a list to store the Products retrieved from DB

        List<Product> products = new ArrayList<>();

        Statement statement = connection.createStatement();

        ResultSet resultSet = statement.executeQuery("SELECT * FROM
PRODUCT");

        //Object relational mapping : converting data returned as table
to objects

        while(resultSet.next()){

            Product product = new Product();

            product.setId(resultSet.getLong("id"));

            product.setName(resultSet.getString("name"));

            product.setPrice(resultSet.getDouble("price"));

            product.setQuantity(resultSet.getInt("quantity"));

            //Adding the product to the list

            products.add(product);

        }

        //returning the list of products

        return products;

    }

    @Override

    public void add(Product o) throws SQLException {

        //Connecting to database

        Connection connection = SingletonDB.getConnection();

        PreparedStatement statement = connection.prepareStatement("INSERT
INTO PRODUCT(name, price, quantity) WHERE (?, ?, ?)");

```

```

        statement.setLong(0, o.getId());

        statement.setString(0, o.getName());

        statement.setDouble(0, o.getPrice());

        statement.setInt(0, o.getQuantity());

        statement.executeUpdate();

    }
}

```

→ IDaoProductImplV2.java

```

package ma.enset.dao;

import ma.enset.entities.Product;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

//Second Implementation of the IDaoProduct interface in which data is
//retrieved from a web service

public class IDaoProductImplV2 implements IDaoProduct{

    @Override

    public List<Product> getData() throws SQLException {

        //Supposed that data is retrieved from REST API (WEB SERVICE)

        List<Product> products = new ArrayList<>();

        products.add(new Product(12, "MACBOOK pro m2", 12000, 4));

        products.add(new Product(12, "PC HP G8", 5600, 14));

        return products;

    }
}

```

```

@Override

public void add(Product o) throws SQLException {

}

}

```

→ SingletonDB.java : Connection à la base de données

```

package ma.enset.dao;

import java.sql.Connection;

import java.sql.DriverManager;

public class SingletonDB {

    private static Connection connection;

    static {

        try {

            String url = "jdbc:mysql://localhost:3306/db2";

            String username="root";

            String passwd="";

            Class.forName("com.mysql.cj.jdbc.Driver");

            connection = DriverManager.getConnection(url, username,
passwd);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

```

```

public static Connection getConnection(){

    return connection;

}

}

```

- La couche metier

→ IMetierProduct.java

```

package ma.enset.metier;

import java.sql.SQLException;

public interface IMetierProduct {

    double calcul() throws SQLException;

}

```

→ IMetierImpl.java

```

package ma.enset.metier;

import ma.enset.dao.IDaoProduct;
import ma.enset.entities.Product;
import java.sql.SQLException;
import java.util.List;

public class IMetierImpl implements IMetierProduct{

    private IDaoProduct iDaoProduct;

    public IMetierImpl() {

    }

    public IMetierImpl(IDaoProduct iDaoProduct) {

```



```
        this.iDaoProduct = iDaoProduct;

    }

    //calculing the average price of products

    @Override

    public double calcul() throws SQLException {

        List<Product> products = iDaoProduct.getData();

        double sumPrice=
products.stream().mapToDouble(Product::getPrice).average().getAsDouble(
);

        return sumPrice;

    }

    public void setiDaoProduct(IDaoProduct iDaoProduct) {

        this.iDaoProduct = iDaoProduct;

    }

}
```

- La couche présentation

→ Pres1.java : Instanciation statique

```
package ma.enset.presentation;

import ma.enset.dao.IDaoProduct;

import ma.enset.dao.IDaoProductImplV1;

import ma.enset.metier.IMetierImpl;

import ma.enset.metier.IMetierProduct;

import java.sql.SQLException;

public class Pres2 {

    public static void main(String[] args) throws SQLException {

        //Instanciation statique en utilisant "new" des objets
        iDaoProductImplV1 et iMetierProduct

        IDaoProduct iDaoProduct = new IDaoProductImplV1();

        //Injection de l'objet iDaoProductImplV1 à l'objet
        iMetierProduct par le constructeur

        IMetierProduct iMetierProduct = new IMetierImpl(iDaoProduct);

        //        //Injection de l'objet iDaoProductImplV1 à l'objet
        iMetierProduct par le setter

        //        IMetierProduct iMetierProduct = new IMetierImpl();

        //        ((IMetierImpl) iMetierProduct).setIDaoProduct(iDaoProduct);

        System.out.println("Version : "+iDaoProduct.getClass()+" ***
Result : "+iMetierProduct.calcul());

    }

}
```

Pres2.java : Instanciation dynamique

```
package ma.enset.presentation;
import ma.enset.dao.IDaoProduct;
import ma.enset.metier.IMetierProduct;
import java.io.File;
import java.lang.reflect.Method;
import java.sql.SQLException;
import java.util.Scanner;
public class Pres1 {
    public static void main(String[] args) throws SQLException {
        //Instanciation dynamique en utilisant un fichier
de configuration config.txt
        try {
            Scanner scanner = new Scanner(new File("config.txt"));
            String daoClassName = scanner.next();
            String metierClassName = scanner.next();
            Class cDao = Class.forName(daoClassName);
            IDaoProduct iDaoProduct = (IDaoProduct)
cDao.getConstructor().newInstance();
            Class cMetier = Class.forName(metierClassName);
            IMetierProduct iMetierProduct = (IMetierProduct)
cMetier.getConstructor().newInstance();
            Method method = cMetier.getMethod("setIDaoProduct",
IDaoProduct.class);
            method.invoke(iMetierProduct, iDaoProduct);
            System.out.println("Version :
"+iDaoProduct.getClass()+" *** Result :
"+iMetierProduct.calcul());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- L'injection des dépendances en utilisant le framework Spring

→ La version XML

Config.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="dao" class="ma.enset.dao.IDaoProductImplV2"/>
    <bean id="metier" class="ma.enset.metier.IMetierImpl">
        <!--Injection by a setter-->
        <!-- <property name="iDaoProduct" ref="dao"/> -->
        <!--Injection by the constructor-->
        <constructor-arg ref="dao"/>
    </bean>
</beans>
```

presSprigXml.java:

```
package ma.enset.presentation;
import ma.enset.metier.IMetierProduct;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext
;
import java.sql.SQLException;
public class presSprigXml {
    public static void main(String[] args) throws SQLException {
        ApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
        IMetierProduct metier = (IMetierProduct)
context.getBean("metier");
        System.out.println(metier.calcul());
    }
}
```

→ La version Annotation

```
//First Implementation of IDaoProduct interface in which data  
@Repository("dao1")  
public class IDaoProductImplV1 implements IDaoProduct{  
    ⚡ //Retreiving data from DB
```

```
//Second Implementation of the IDaoProduct interface in which data  
@Repository("dao2")  
public class IDaoProductImplV2 implements IDaoProduct{  
    1 usage
```

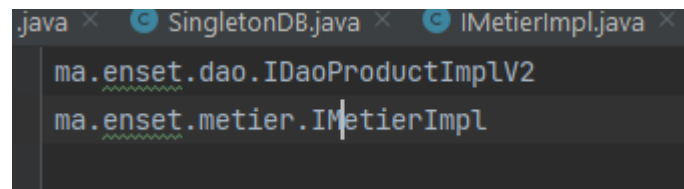
```
@Service  
public class IMetierImpl implements IMetierProduct{  
    3 usages  
    private IDaoProduct iDaoProduct;  
  
    //    public IMetierImpl() {  
    //    }  
  
    public IMetierImpl(@Qualifier("dao2") IDaoProduct iDaoProduct) { this.iDaoProduct = iDaoProduct; }
```

presSpringAnnotation.java:

```
package ma.enset.presentation;  
  
import ma.enset.metier.IMetierProduct;  
import org.springframework.context.ApplicationContext;  
import  
org.springframework.context.annotation.AnnotationConfigApplicationConte  
xt;  
import java.sql.SQLException;  
  
public class presSpringAnnotation {  
    public static void main(String[] args) throws SQLException {  
        ApplicationContext context = new  
AnnotationConfigApplicationContext("ma.enset.metier", "ma.enset.dao");  
        IMetierProduct metier = context.getBean(IMetierProduct.class);  
        System.out.println(metier.calcul());  
    }  
}
```

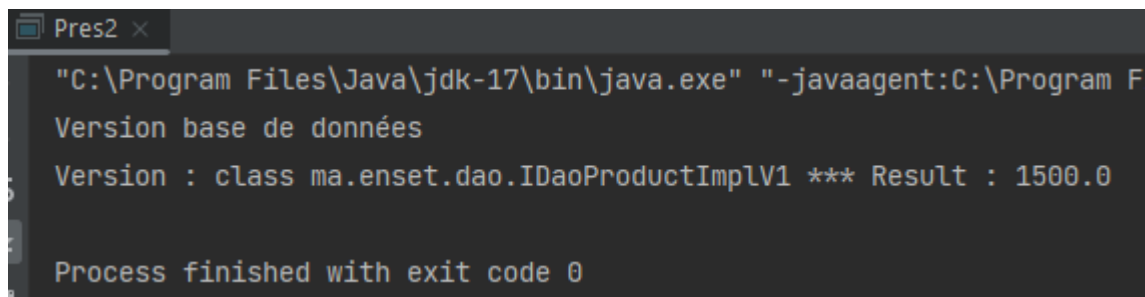
3) Captures d'écran

- Fichier de configuration pour l'instanciation dynamique



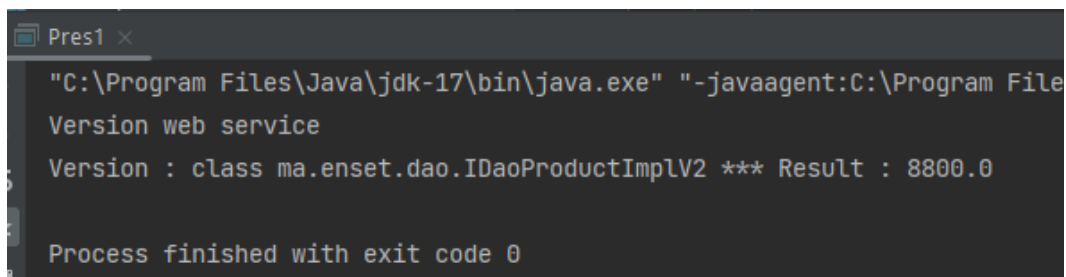
```
.java x SingletonDB.java x IMetierImpl.java x
ma.enset.dao.IDaoProductImplV2
ma.enset.metier.IMetierImpl
```

- Version base de données



```
Pres2 x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program F
Version base de données
Version : class ma.enset.dao.IDaoProductImplV1 *** Result : 1500.0
Process finished with exit code 0
```

- Version web service



```
Pres1 x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program File
Version web service
Version : class ma.enset.dao.IDaoProductImplV2 *** Result : 8800.0
Process finished with exit code 0
```

Conclusion

En conclusion, ce TP a permis une exploration approfondie des méthodes d'**injection de dépendances**, mettant en lumière les principes fondamentaux de l'**instanciation statique et dynamique**. À travers cette expérience, nous avons pu constater l'importance du **couplage faible** dans la conception de **logiciels flexibles et évolutifs**. En favorisant la dépendance aux **interfaces** plutôt qu'aux **classes concrètes**, nous avons pu rendre notre programme **fermé à la modification et ouvert à l'extension**. L'approche adoptée lors de ce TP, en débutant par l'instanciation statique et en poursuivant avec l'instanciation dynamique, a permis une compréhension approfondie des mécanismes sous-jacents et de leurs implications dans la conception d'architectures JEE et de middleware. En somme, ce compte rendu témoigne de notre engagement à appliquer les principes d'architecture logicielle de manière rigoureuse, tout en soulignant l'importance du couplage faible comme pilier fondamental de la flexibilité et de l'évolutivité des systèmes informatiques.