TP1 : Inversion de Contrôle (IoC) et Injection de Dépendances avec Spring

Objectifs pédagogiques

À l'issue de ce TP, l'étudiant sera capable de :

- Expliquer le principe d'Inversion de Contrôle (IoC)
- Configurer un conteneur Spring avec XML
- Utiliser l'injection par constructeur et par setter
- Configurer Spring avec des annotations
- Comparer XML vs annotations
- Comprendre les bonnes pratiques actuelles

Prérequis

- Java 8+
- Maven ou Gradle
- Notions de base en Java (POO)
- IDE (IntelliJ / Eclipse)

Contexte du TP

On développe une **application simple de gestion de notifications**.

- Un service métier envoie des notifications
- Le canal de notification est interchangeable (Email, SMS)
- Le choix de l'implémentation est délégué à Spring

Objectif : **ne jamais instancier les dépendances avec `new`**

Partie 1 – Sans Spring (problème initial)

1.1 Implémentation naïve

```
public class NotificationService {

    private EmailSender sender = new EmailSender();

    public void notifyUser(String msg) {
        sender.send(msg);
    }
}
```

Questions

1. Où est le couplage fort ?
2. Peut-on facilement remplacer `EmailSender` ?
3. Ce code respecte-t-il l'IoC ?

Partie 2 – IoC avec Spring et XML

2.1 Création des interfaces

```
public interface MessageSender {
    void send(String message);
}

public class EmailSender implements MessageSender {
    public void send(String message) {
        System.out.println("Email : " + message);
    }
}

public class SmsSender implements MessageSender {
    public void send(String message) {
        System.out.println("SMS : " + message);
    }
}
```

2.2 Service métier

```
public class NotificationService {

    private MessageSender sender;

    public NotificationService(MessageSender sender) {
```

```
        this.sender = sender;
    }

    public void notifyUser(String msg) {
        sender.send(msg);
    }
}
```

2.3 Configuration Spring XML

Créer `applicationContext.xml`

```
<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="emailSender" class="com.tp.EmailSender"/>

    <bean id="notificationService"
          class="com.tp.NotificationService">
        <constructor-arg ref="emailSender"/>
    </bean>

</beans>
```

2.4 Test de l’application

```
ApplicationContext context =
        new ClassPathXmlApplicationContext("applicationContext.xml");

NotificationService service =
        context.getBean(NotificationService.class);

service.notifyUser("Bonjour Spring");
```

Questions

1. Qui crée les objets ?
2. Où est l’IoC ?
3. Comment changer le canal de notification ?

2.5 Injection par setter (variante)

Modifier `NotificationService` :

```
public class NotificationService {

    private MessageSender sender;

    public void setSender(MessageSender sender) {
        this.sender = sender;
    }

    public void notifyUser(String msg) {
        sender.send(msg);
    }
}
```

XML :

```
<property name="sender" ref="smsSender"/>
```

Comparaison attendue

- Injection constructeur vs setter
- Avantages / inconvénients

Partie 3 – IoC avec annotations

3.1 Activation du component scanning

```
<context:component-scan base-package="com.tp"/>
```

3.2 Annotations sur les composants

```
@Component
public class EmailSender implements MessageSender {
    public void send(String message) {
        System.out.println("Email : " + message);
    }
}

@Service
public class NotificationService {

    private final MessageSender sender;

    @Autowired
    public NotificationService(MessageSender sender) {
        this.sender = sender;
    }

    public void notifyUser(String msg) {
```

```
        sender.send(msg);
    }
}
```

3.3 Gestion des ambiguïtés

Ajouter `SmsSender` :

```
@Component
public class SmsSender implements MessageSender {
    public void send(String message) {
        System.out.println("SMS : " + message);
    }
}
```

Problème : plusieurs implémentations

Solution :

```
@Autowired
public NotificationService(@Qualifier("emailSender") MessageSender sender) {
    this.sender = sender;
}
```

3.4 Test avec annotations

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("applicationContext.xml");

NotificationService service =
    context.getBean(NotificationService.class);

service.notifyUser("Injection réussie");
```

Partie 4 – Comparaison XML vs Annotations

Tableau à compléter

Critère	XML	Annotations
Lisibilité		
Couplage		
Refactoring		
Configuration dynamique		

Partie 5 – Bonnes pratiques (obligatoire)

1. Préférer l’injection par **constructeur**
2. Éviter l’injection par champ
3. Utiliser :
    - `@Repository` pour la persistance
    - `@Service` pour le métier
    - `@Controller` pour le web
4. Réserver XML à l’infrastructure

Partie 6 – Questions de réflexion (examen)

1. En quoi Spring implémente-t-il l’IoC ?
2. Quelle différence entre IoC et DI ?
3. Quel pattern GoF est utilisé implicitement ?
4. Pourquoi Spring favorise-t-il le constructor injection ?

Livrables attendus

- Code source fonctionnel
- Réponses aux questions
- Comparaison XML vs annotations
- Schéma UML simple montrant les dépendances
