MSDIA 2024-2025

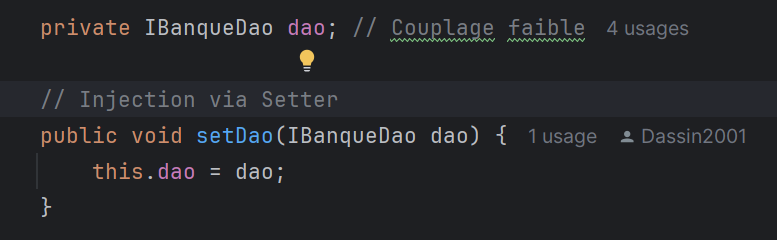
Partie 2 : Mini Projet

AAMER FADMA

**1- A travers un fichier XML de configuration en utilisant Jax Binding (OXM : Mapping Objet XML)**

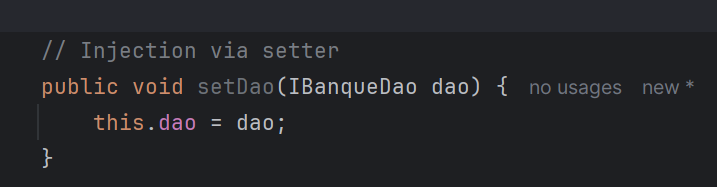
Nous allons écrire un fichier XML de configuration simulant ce que ferait Spring. Ensuite, on va parser ce XML pour créer les objets, et injecter la dépendance DAO dans la couche métier selon 3 styles différents.

**a- Le constructeur**

****

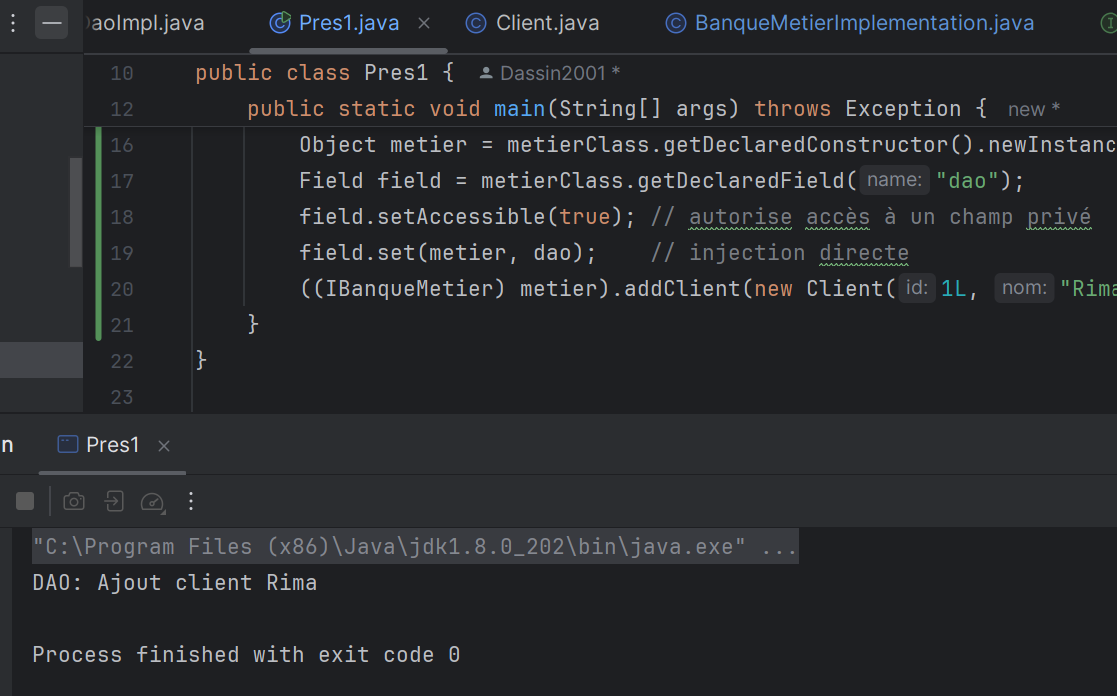
**b- Le Setter**

****

****

**c- Attribut (accès direct à l'attribut : Field)**

Cette méthode injecte la dépendance dao directement dans le champ privé de la classe métier, sans passer par un setter ni un constructeur.

****

**2- En utilisant les annotations**

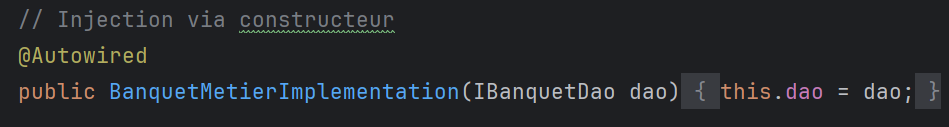
**Dans cette deuxième approche, on utilise le conteneur Spring et ses annotations pour détecter automatiquement les classes, les instancier, et injecter les dépendances.**

**On indique à Spring de scanner le package "rima.com". Il va détecter automatiquement les classes annotées, créer les objets nécessaires et injecter les dépendances.**

****

1. **Le constructeur**

Dans cette méthode, nous allons injecter la dépendance via un constructeur annoté avec @Autowired. Cette approche est fortement recommandée car elle garantit que la dépendance est obligatoire pour le bon fonctionnement de l’objet.

* **Pour cela, on doit :**
* Annoter la classe métier avec @Service pour la rendre visible à Spring.
* Déclarer un constructeur qui accepte la dépendance en paramètre.
* Annoter ce constructeur avec @Autowired.

****

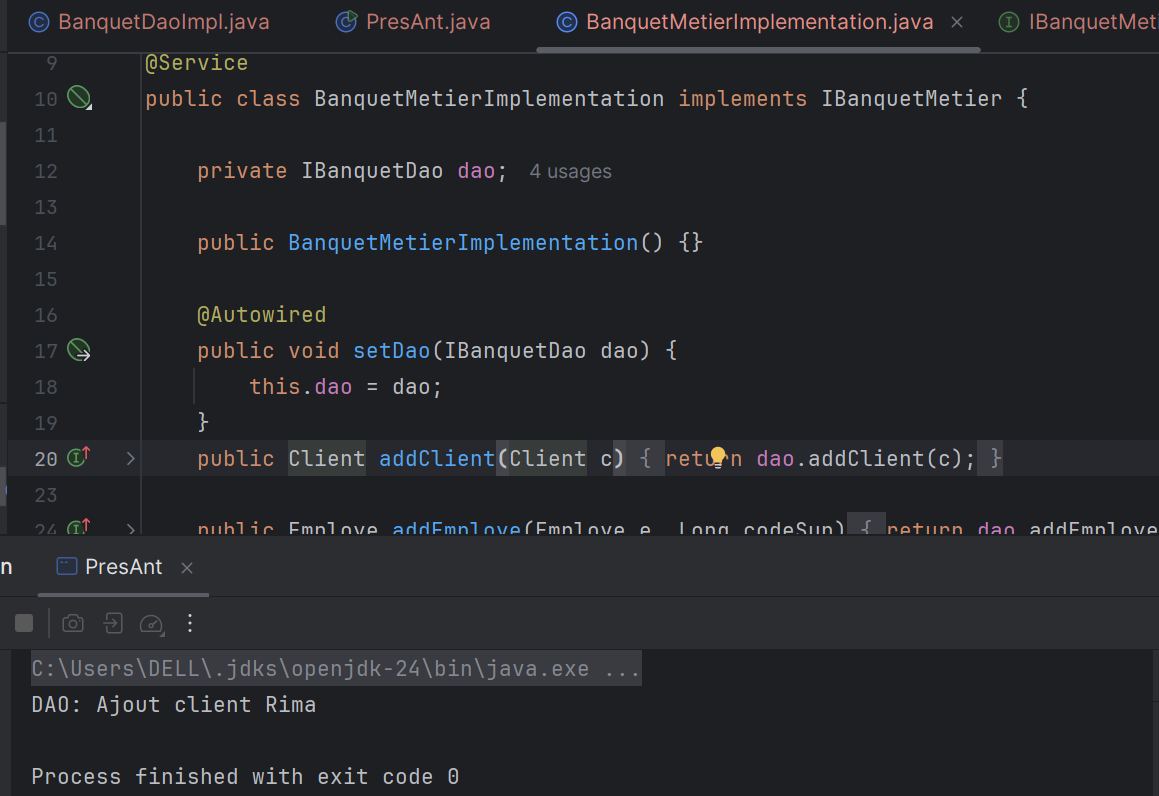
Grâce à @Autowired sur le constructeur, Spring injecte automatiquement un objet de type IBanqueDao. Le champ dao est déclaré final pour garantir qu’il est toujours initialisé (bonne pratique).

1. **Le Setter**

Nous allons maintenant injecter la dépendance en utilisant une **méthode setter** annotée avec @Autowired. Cette technique est plus flexible car la dépendance peut être définie ou modifiée après la création de l’objet.

### Pour cela, on doit :

* Fournir un constructeur vide (ou ne rien déclarer si aucun constructeur n'est défini).
* Définir un **setter public** pour la dépendance.
* Annoter ce setter avec @Autowired.

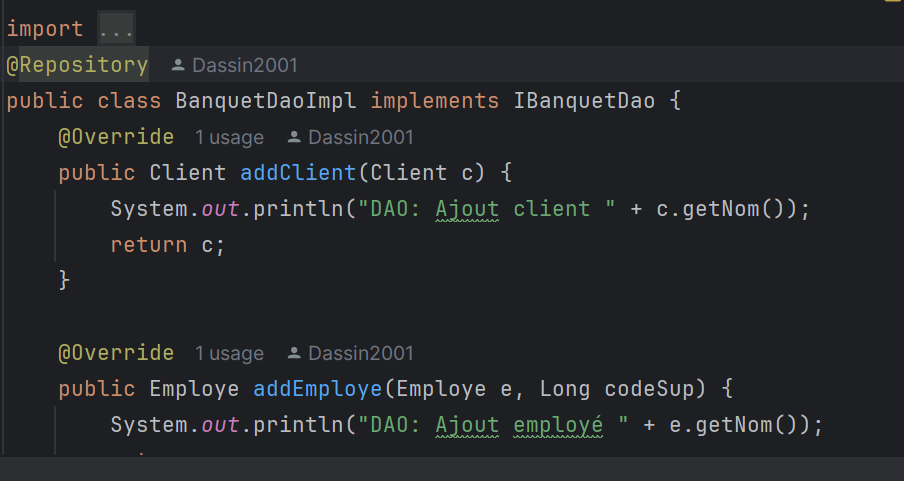


Avec cette approche, Spring crée l’objet sans paramètre, puis appelle la méthode setDao() en lui injectant une instance de IBanqueDao. **c- Attribut (accès direct à l'attribut : Field)**

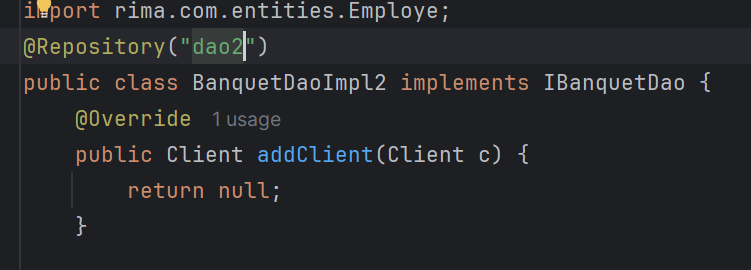
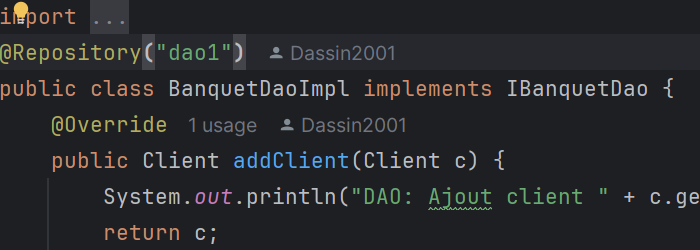
Enfin, nous allons réaliser l’injection de dépendance directement dans un attribut privé, sans utiliser ni setter ni constructeur. Cette méthode est la plus concise, mais elle est moins adaptée au test unitaire car les champs sont généralement inaccessibles sans réflexion.

**Pour cela, on doit :**

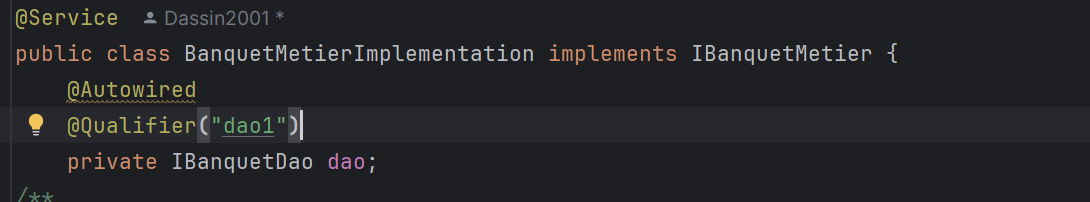
* Déclarer un champ privé de type IBanqueDao.
* Annoter ce champ avec @Autowired.
* Spring identifie le champ annoté et lui affecte automatiquement une instance du bean adéquat lors de l’instanciation de l’objet métier.
* Pour permettre à Spring d’injecter automatiquement l’implémentation de l’interface IBanqueDao, nous devons également annoter la classe DAO avec @Repository. Cela permet à Spring de reconnaître cette classe comme un composant de la couche d’accès aux données.

****

* Dans le cas où plusieurs classes implémentent la même interface, Spring ne peut pas savoir laquelle injecter. Pour résoudre ce problème, on utilise l’annotation @Qualifier afin de préciser le nom du bean à injecter



* Dans la classe métier :



**Conclusion**

Grâce à l’approche basée sur les annotations, l’injection des dépendances devient plus lisible, plus légère, et ne nécessite plus de configuration XML fastidieuse. Pour cela, on utilise des annotations simples comme @Component, @Service, @Autowired et @Qualifier.  
Cette méthode permet de respecter le **principe de faible couplage** tout en rendant le code plus modulaire, plus testable et plus facile à maintenir.