Architecture JEE :

Principe de l'Inversion de Contrôle et Injection de dépendances

Remarque :

Il est très difficile de développer un système logiciel qui respect ces exigences sans utiliser l'expérience des autres :

- Bâtir l'app sur une architecture d'entreprise : JEE

- **Framework pour l'inversion de contrôle :**
  - **Permettre au développeur de s'occuper uniquement du code métier (Exigences fonctionnelles) et c'est le Framework qui s'occupe du code technique (Exigences Techniques)**
  - **Dans L'architecture JEE le Framework le plus utilisé pour l'inversion de contrôle c'est le Framework Spring IOC**
  - **Spring IOC permet d'intégrer facilement les autres Framework pour gérer les aspects techniques**
    - JPA, Hibernate Pour le mapping Object relationnel
    - Spring MVC pour la partie Web de l'application
    - Spring Security pour la sécurité
    - Spring Transactionnel pour la gestion des transactions

Exemple : sans utiliser d'inversion de contrôle

```
public void virement(int c1, int c2, double mt) {
    /* Création d'une transaction */
        EntityTransaction transaction=entityManager.getTransaction();
    /* Démarrer la transaction */
        transaction.begin();
        try {                                                    Code Technique
            /* Code métier */
            retirer(c1,mt);
            verser(c2,mt);                                       Code Métier
            /* Valider la transaction  */
            transaction.commit();
        } catch (Exception e) {
          /* Annuler la transaction en cas d'exception          Code Technique
            transaction.rollback();
            e.printStackTrace();
        }
}
```

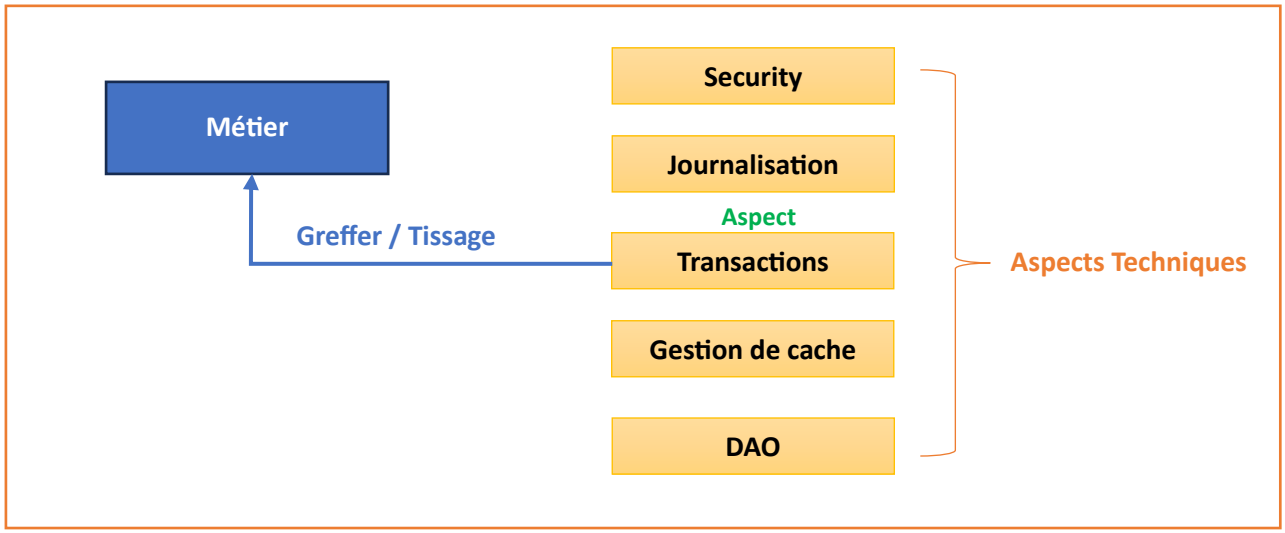
Exemple : en utiliser l'inversion de contrôle

```
@Transactional
public void virement(int c1, int c2, double mt) {
        retirer(c1,mt);
        verser(c2,mt);
}
```

Code Métier

- **Avec l'annotation @Transactional, le développeur se concentre uniquement sur le code métier de l'app et le Framework ici Spring IOC se concentre sur le code technique.**

- **Grace au paradigme Programmation Orienté Aspect (AOP) on peut séparer des aspects métiers et des aspects techniques d'une app.**



Rappel de quelques principes de conception :

- Une app qui n'évolue pas meurt.
- Une app doit être fermé à la modification et ouverte à l'extension
- Une app doit s'adapter aux changements

Alors :

- **Comment Créer une app fermé à la modification et ouverte à l'extension ?**
- **Comment faire l'injection des dépendances ?**
- **C'est quoi le principe de l'inversion de contrôle ?**