Inversion de Contrôle (IoC) et Injection de Dépendances (DI)

7. Comprendre l'Injection de Dépendances (Dependency Injection - DI)

Pour maîtriser Spring, il est crucial de comprendre son cœur fonctionnel : la gestion des objets. C'est ici qu'entrent en jeu les concepts d'**Inversion de Contrôle (IoC)** et d'**Injection de Dépendances (DI)**.

Qu'est-ce que l'Inversion de Contrôle (IoC) ?

Dans la programmation orientée objet traditionnelle, votre code contrôle le flux d'exécution et instancie lui-même les objets dont il a besoin (via le mot-clé `new`). Avec l'**Inversion de Contrôle**, vous déléguez cette responsabilité au framework (le conteneur Spring). C'est le framework qui gère le cycle de vie des objets (création, configuration, destruction) et qui appelle votre code au bon moment.

Qu'est-ce que l'Injection de Dépendances (DI) ?

L'Injection de Dépendances est la technique utilisée par Spring pour mettre en œuvre l'IoC. Si une classe A a besoin d'une classe B pour fonctionner, la classe B est une "dépendance" de la classe A. Au lieu que la classe A instancie la classe B, le conteneur Spring va *injecter* (fournir) une instance de B directement dans A.

> **Pourquoi est-ce indispensable ?** Cela rend votre code *faiblement couplé* (Loosely Coupled). Il devient beaucoup plus facile à tester unitairement, à maintenir, et vous n'avez plus à gérer manuellement les dépendances en cascade.

8. Comment Spring gère-t-il les objets ? (@Component et @Autowired)

Spring utilise des annotations pour repérer les classes qu'il doit gérer et pour lier ces classes entre elles. Dans le vocabulaire de l'écosystème Spring, un objet instancié et géré par le conteneur est appelé un **Bean**.

apcpedagogie.com

- `@Component`: Placée au-dessus d'une classe, cette annotation indique à Spring : "*Prends le contrôle de cette classe, instancie-la (crée un Bean) et garde-la dans ton contexte (mémoire).*" *Note : Spring propose des spécialisations sémantiques de `@Component` comme `@Service` (pour la logique métier), `@Repository` (pour l'accès aux données) et `@Controller` (pour les API Web).*
- `@Autowired`: Placée sur un constructeur ou un champ, elle indique à Spring : "*Cette classe a besoin d'une dépendance ici. Cherche le Bean correspondant dans ton contexte et injecte-le automatiquement.*"

Exemple pratique : Application au traitement narratif

Imaginons l'architecture d'un jeu d'investigation narratif. Nous avons besoin d'un service pour analyser les indices récoltés sur une scène de crime, et d'un contrôleur Web pour gérer l'avancée du dossier.

1. Le Service (La dépendance) :
Nous créons un service contenant la logique de déduction. Nous utilisons `@Service` pour le déclarer comme Bean Spring.

```
package com.studiogames.mysterybackend.service;

import org.springframework.stereotype.Service;

// Spring va créer et gérer une instance unique (Singleton) de cette classe
@Service
public class EvidenceAnalyzerService {

    public String analyzeClue(String clue) {
        // Logique d'analyse de l'indice
        return "Analyse de l'indice : " + clue + ". La logique de déduction pointe vers l'un
des 2 suspects principaux.";
    }
}
```

2. Le Contrôleur (L'endroit où la dépendance est injectée) :
Au lieu d'écrire `new EvidenceAnalyzerService()`, nous laissons Spring injecter le service via le constructeur. C'est la méthode recommandée (Constructor Injection).

```
package com.studiogames.mysterybackend.controller;

import com.studiogames.mysterybackend.service.EvidenceAnalyzerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
@RequestMapping("/api/cases/the-vanishing-diplomat")
public class DiplomatCaseController {

    // Déclaration de la dépendance (en utilisant 'final' pour s'assurer qu'elle ne change
pas)
    private final EvidenceAnalyzerService evidenceAnalyzer;

    // L'annotation @Autowired sur le constructeur réalise l'Injection de Dépendances.
    // Spring va automatiquement fournir l'instance d'EvidenceAnalyzerService lors de la
création du contrôleur.
    // (À noter : Depuis Spring 4.3, si une classe n'a qu'un seul constructeur, @Autowired
est implicite et peut être omis).
    @Autowired
    public DiplomatCaseController(EvidenceAnalyzerService evidenceAnalyzer) {
        this.evidenceAnalyzer = evidenceAnalyzer;
    }

    @GetMapping("/inspect")
    public String inspectCrimeScene() {
        String foundClue = "Notes dactylographiées avec des accents d'encre ambre";

        // Nous utilisons le service injecté par Spring sans l'avoir instancié manuellement
        return evidenceAnalyzer.analyzeClue(foundClue);
    }
}
```

L'énorme avantage de cette approche :

Si demain vous décidez de modifier la façon dont les indices sont gérés (par exemple, en connectant `EvidenceAnalyzerService` à une base de données Cloud Firestore au lieu de générer du texte statique), vous n'aurez absolument **rien à changer** dans le `DiplomatCaseController`. Vous modifiez uniquement le service, et Spring se charge de distribuer la nouvelle version mise à jour à tous les composants qui en dépendent. C'est la beauté du code faiblement couplé !

Rédigé et structuré par : **Med Khalil Kribi**

Source : https://apcpedagogie.com/ioc-et-injection-dependances-spring · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com