*Simplifier le développement d’application Java avec Spring*

**Chapitre 1 : Mettre en œuvre une inversion de contrôle**

Spring est un framework qui vient rendre de la souplesse à l’architecture de nos applications notamment grâce à l’injection de dépendance ou l’inversion de contrôle qu’on peut appliquer selon les principes SOLID. Spring n’est pas une solution « tout ou rien » mais organisée sous formes de « modules ».

1. **Architecture et organisation :**

Le projet de ce cours est organisé en 5 modules :

* Module ticket-**batch**
  + Contient les batchs de l’appli java
* Module ticket-**webapp**
  + Contient l’appli web (ici simple API REST sans front-end)
* Module ticket-**business**
  + Contient la logique métier de l’appli Java (les servlets)
  + Un servlet, ou ici manager, pour chaque sous-packages des beans
* Module ticket-**consumer**
  + Contient les éléments d’interaction avec l’environnement extérieur (par exemple les DAO)
* Module ticket-**model**
  + Contient les objets métiers (les beans)
  + Les objets models sont divisés en trois autres packages (bean, recherche et exception)

Il existe en réalité un sixième module, appelé ticket-technical, qui contient les éléments et dépendance techniques de l’appli java (comme les logs), et dont tous les autres modules dépendent.

1. **Dessine-moi une inversion de contrôle (IoC) :**

Les principes SOLID sont 5 principes de bases sur lesquelles se repose la programmation orientée objets et représentés chacun par l’une des lettres de l’acronyme. Les lettres S et D représente les principes suivants : « Single responsability » et « Independence inversion ». Cela implique qu’un module de plus haut niveau ne doit pas dépendre d’un module de plus bas niveau pour fonctionner (les deux doivent dépendre d’abstraction) et que chaque module (ou classe) doit avoir un rôle et une responsabilité unique.

Dans notre cas, la classe ProjetRessource instancie la classe ProjetManager (qui se trouve dans le module business) car elle a besoin d’elle pour fonctionner. Ainsi, il n’est pas logique qu’une classe du module webapp se charge d’instancier une classe d’un module plus haut niveau (on ne demande pas au cuisinier de fabriquer ses plats, on lui en fournit un) et il vaudrait mieux pouvoir directement appeler une instance de la classe ProjectManager sans se soucier de son instanciation.

C’est là qu’entre en jeux le principe de l’inversion de contrôle. L’inversion de contrôle n’est en fait qu’un terme générique qui indique que le flot d’exécution d’un logiciel n’est plus sous le contrôle direct de l’application elle-même mais sous celui de sa couche logicielle sous-jacente (la plupart du temps un framework). Elle peut se matérialiser sous différentes formes, notamment la fameuse injection de dépendances ou une factory (voir un singleton) qui viendront s’occuper d’instancier les bons objets au bons moments.

Define « factory » : Une *factory est un patron de conception qui permet d’instancier des objets dont le type est dérivé d’un type abstrait. La classe exacte de l’objet n’est donc pas connue par l’appelant. Un programme n’ayant en général besoin que d’une seule factory, on utilise souvent le patron de conception singleton*

Define « singleton » : *Le signleton a la particularité d’envoyer toujours la même instance d’un objet là où la factory simple enverra à chaque fois une nouvelle instance d’un objet.*

Texte…