JEE
LE FRAMEWORK SPRING

LICENCE PRO. DAWIN

H. BERGER M.A. TESSIER

2022-2023

INTRODUCTION AU FRAMEWORK SPRING

- Le **framework Spring** permet de réaliser des **applications JEE** dans leur **globalité** afin que celles-ci soient organisées de manière structurée.
- Il fournit de **nombreuses fonctionnalités** parfois redondantes ou qui peuvent être configurées ou **utilisées de plusieurs manières** : ceci **laisse le choix au développeur** d'utiliser la solution qui lui convient le mieux et/ou qui répond aux besoins.

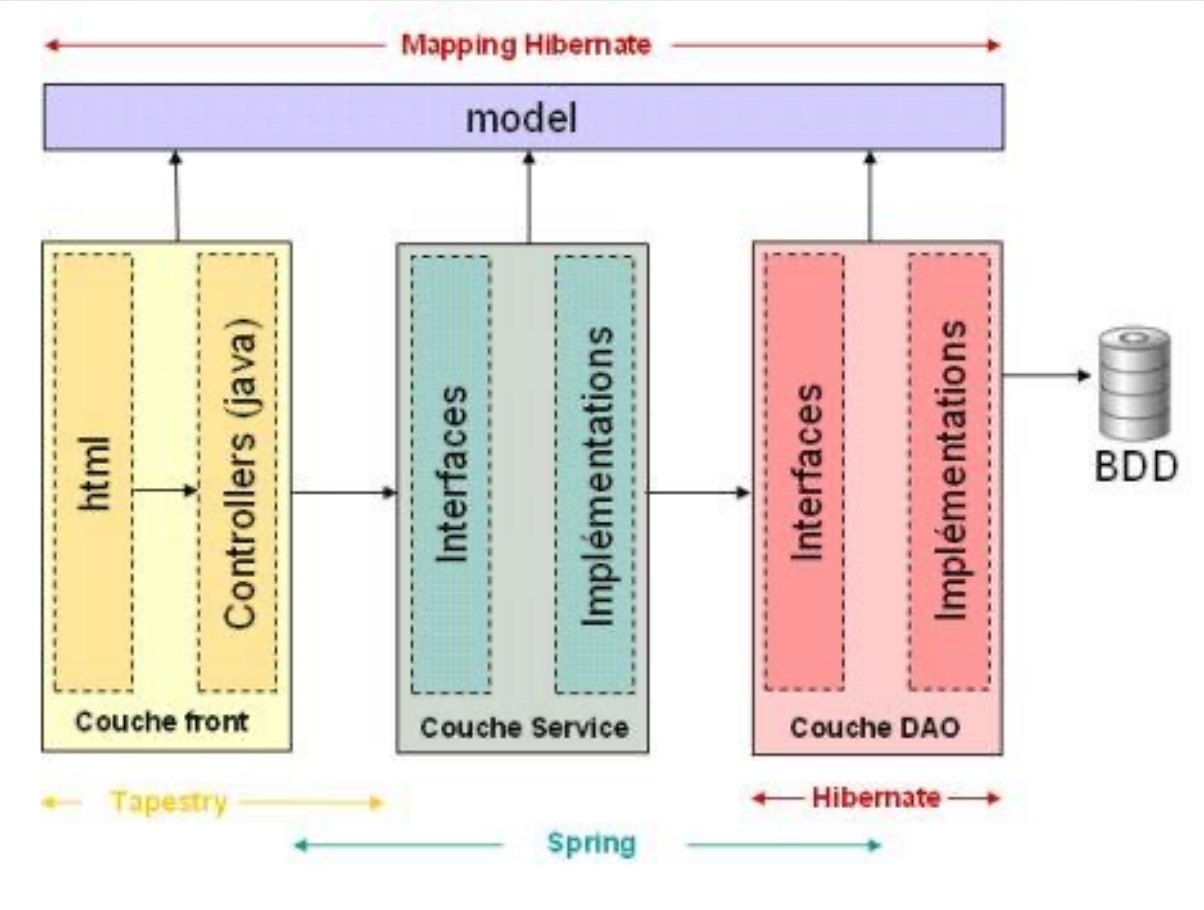
INTRODUCTION AU FRAMEWORK SPRING

- **conteneur de type IoC** assurant la gestion du **cycle de vies des beans** et **l'injection des dépendances**
- utilisation de **l'AOP (Aspect Oriented Programming)**

→ Objectif : externaliser les fonctionnalités transverses (logs, habilitations, gestion des transactions…)sous la forme d'aspects.

- facilite l'intégration avec de nombreux **projets open source** ou **API** de Java EE
- favorise l'intégration avec de nombreux autres **frameworks** notamment ceux de type **ORM** ou **web** (exemple: Hibernate, Struts…)

ARCHITECTURE APPLICATIVE FRAMEWORK SPRING



SPRING: LES REPOSITORIES

- Le framework Spring gère l'équivalent des **DAO** (Data Access Object) à l'aide de **repositories**.
- Chaque repository possède une **interface** (liste de **signatures** de **méthodes**).
- Le framework Spring propose ensuite différentes possibilités pour l'implémentation des interfaces de repository. Exemples:
  - implémentation à l'aide du **standard JPA**
  - Implémentation à l'aide de **JDBC**
  - Implémentation à l'aide de **SpringDataJPA** (spécifique à Spring)
- La couche service fera ensuite appel aux repositories en passant par les **interfaces de repositories**, sans se soucier de **la manière dont elles ont été implémentées** (principe d'abstraction respecté).
- L'avantage est des repositiries est de pouvoir proposer de manière transparente des implémentations vers **différents types de SGBD** (relationnels ou NoSQL).
- Le type d'implémentation choisi est indiqué dans les **fichiers de configuration** de l'application JEE (fichiers .xml)

LES 5 PRINCIPES SOLID

Pour développer une application de qualité et structurée, il est conseillé de respecter les **5 principes SOLID**:

- **Single Responsability Principle (SRP)** (responsabilité : raison de changer) : une **seule responsabilité par classe** pour permettre d'isoler le changement et être sur qu'en cas de changement on impactera pas d'autres responsabilités.
- **Open Close Principle** : un **programme doit être fermé à la modification, ouvert à l'extension** pour **ne pas changer le code interne** au composant mais avoir des points d'extensions (principe émis par le français Bertrand Mayer). Pour ajouter un comportement il ne faut pas avoir a toucher à l'existant.
- **Liskov substitution principle** : Si on fournit une abstraction en retournant une **interface**, on l'utilise sans que ça la casse. Si quelqu'un utilise une **classe mère**, on peux ajouter une ou des classe(s) fille sans casser la classe mère.

LES 5 PRINCIPES SOLID (SUITE)

- **Interface segregation principle** : préférer **plusieurs interfaces spécifiques** pour chaque client plutôt qu'une **seule interface générale**.
- **Dependency inversion principle** : les *couches de hauts niveaux* ne doivent pas dépendre des **couches de bas niveaux** mais les couches de bas niveaux dépendent des couches de haut niveau. On **supprime donc la dépendance aux outils techniques** dans la modification des programmes.

PRINCIPE DE L'INVERSION DE CONTRÔLE (IoC) avec INJECTION DE DEPENDANCES

- Un exemple pour comprendre:

```
// Interface HelloWorld
public interface HelloWorld {
    public void sayHello();
}

// Class implements HelloWorld
public class SpringHelloWorld
implements HelloWorld {
   public void sayHello()  {
          System.out.println("Spring say
Hello!");
   }
}
```

```
// Other class implements HelloWorld
public class StrutsHelloWorld implements
HelloWorld {
   public void sayHello()  {
          System.out.println("Struts say
Hello!");
   }
}
```

PRINCIPE DE L'INVERSION DE CONTRÔLE (IoC) avec INJECTION DE DEPENDANCES (suite)

- Un exemple pour comprendre (suite):

```
// And Service class
public class HelloWorldService {

    // Field type HelloWorld
    private HelloWorld helloWorld;

    // Constructor HelloWorldService
    // It initializes the values for the field
'helloWorld'
    public HelloWorldService()  {
        this.helloWorld = new
StrutsHelloWorld();
    }
}
```

→Ainsi, HelloWorldService contrôle la *création de l'ojet* HelloWorld.

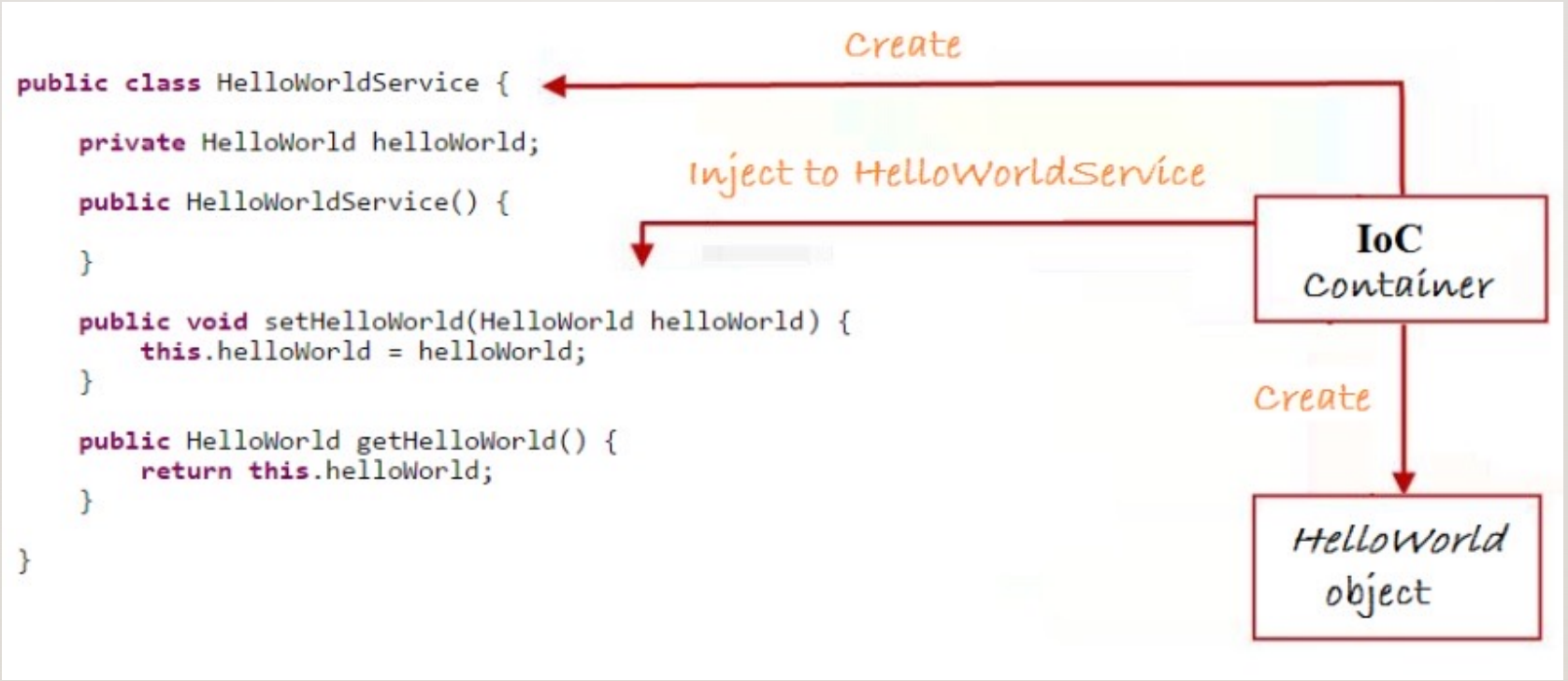
→ Pourquoi ne pas transférer la création de HelloWorld à un tiers, au lieu de le faire dans HelloWorldService?

→ Nous avons la définition de "*inversion of control*" qui signifie "inversion de contrôle" (IoC).

→ L'IoC Container agira en tant que gestionnaire et créera HelloWorldService et *HelloWorld*.

PRINCIPE DE L'INVERSION DE CONTRÔLE (IoC) avec INJECTION DE DEPENDANCES

```java
public class HelloWorldService {

    private HelloWorld helloWorld;

    public HelloWorldService() {

    }

    public void setHelloWorld(HelloWorld helloWorld) {
        this.helloWorld = helloWorld;
    }

    public HelloWorld getHelloWorld() {
        return this.helloWorld;
    }

}
```



LA NOTION D'ABSTRACTION

- Dans les différentes couches de l'application, des **interfaces** sont créées (exemple: les repositories, dans la couche service).
- Ainsi, les différentes couches de l'application pourront invoquer ces interfaces **sans avoir à se soucier de la manière dont elles sont implémentées** (exemple: plusieurs implémentation possibles pour les repositories, avec JPA, JDBC ou SpringDataJPA).

SPRING: POURQUOI UNE COUCHE SERVICE?

- La **couche service** va jouer le rôle de couche intermédiaire entre la couche contrôleur et la couche d'accès aux données (avec les repositories).
- C'est dans la couche service que va se trouver la **logique métier** en implémentant les règles de gestion de l'application.
- Cette couche permet aussi de **fournir une interface** qui **liste tous les services** proposés par l'application.