JAVA/J2EE au plus haut niveau

# SPRING – Travaux pratiques

par SAMIR MILOUDI

# Programme

**TP 1 :** Injection de dépendances et `BeanFactory`

**TP 2 :** Injection automatique des collaborateurs

**TP 3 :** Détection automatique des composants

**TP 4 :** Premiers pas avec Spring MVC

**TP 5 :** Spring MVC – gestion des formulaires

# Programme

**TP 6 :** Spring et JPA

**TP 7 :** Spring et l'AOP

**TP 8 :** Spring Security

Tous les TP sont des projets Maven.

Avant de les importer dans Eclipse, il faut les préparer avec :

```
mvn eclipse:eclipse
```

ecoalis

# TP 1

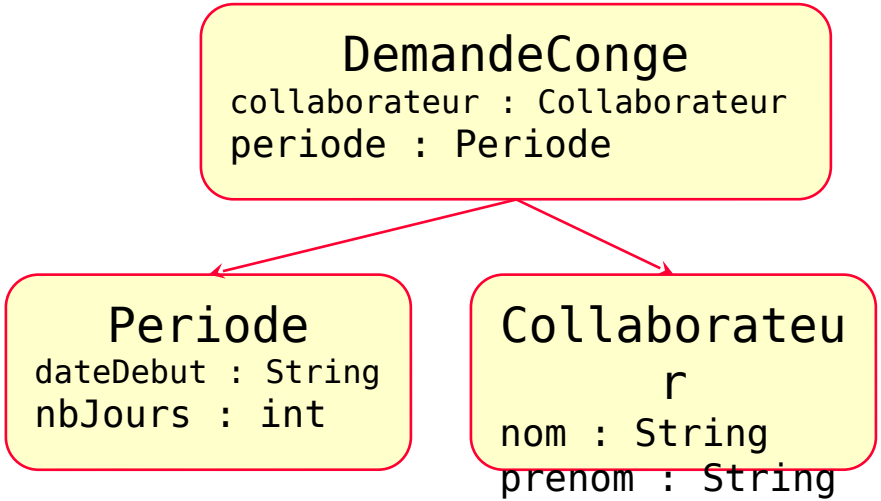
# Injection de dépendances et BeanFactory

## Objectifs

- Déclarer des Beans dans le conteneur Spring
- Comprendre le chargement du conteneur Spring
- Mettre en œuvre l'injection de dépendances

Créer les classes indiquée sur le schéma ainsi que leurs attributs et modificateurs.

Aucun constructeur n'est requis.




Utiliser `org.apache.commons.lang.builder.ToStringBuilder` pour la méthode `toString()`

Dans le fichier de configuration de Spring, ajouter trois Beans correspondant à ces trois classes.

Identifier les Bean :

- `DemandeConge : demandeConge`
- `Collaborateur : c`
- `Periode : p`

Compléter les propriétés des Beans « `Collaborateur c` » et « `Periode p` » avec des valeurs au choix.

## Configurer l'injection de dépendances :

Compléter les propriétés du Bean « `DemandeConge` » en référençant les Beans « `Collaborateur c` » et « `Periode p` ».

# TP1 - Injection de dépendances et BeanFactory

**Compléter la classe de test pour :**

- charger le conteneur Spring : récupérer une instance de « `BeanFactory` » implémentée par « `XmlBeanFactory` »
- récupérer le Bean « `demandeConge` » du conteneur
- afficher ce Bean sur la sortie standard

## Portée des Beans :

Dans la classe de test, récupérer une seconde fois le Bean « demandeConge » (variable « demandeConge2 ») et l'afficher sur la sortie standard.

Que constate t-on ?

**Question :**

Que faut-il ajouter à la définition des Beans pour qu'ils soient instanciés à chaque demande ?

## Injection de dépendances par constructeur :

Créer la classe « `DemandeCongeSansModificateurs` », avec les mêmes attributs que « `DemandeConge` » mais sans modificateurs.

L'injection des dépendances au collaborateur et à la période se fait désormais par **constructeur**.

Ajouter ce Bean au fichier de configuration Spring et y injecter les dépendances requises.

Dans la classe de test, afficher ce Bean sur la sortie standard.

**Question :**

Que se passe t'il si on inverse les références au niveau des « `constructor-arg` » ?

# TP 2

# Injection automatique des collaborateurs

## Objectifs

- Injecter automatiquement les collaborateurs selon les deux méthodes proposées par Spring

## Remarque

« `BeanFactory` » ne permet pas d'utiliser l'autowiring et d'autres fonctionnalités très utiles.

Il est donc toujours recommandé d'utiliser
« `ApplicationContext` » à la place. (cf. classe de test TP2)

## Méthode 1 : autowiring en XML

Modifier le fichier de configuration Spring fourni pour que les collaborateurs des deux demandes de congés y soient injectés automatiquement :

- par « type » pour « `DemandeConge` »
- par « constructeur » pour « `DemandeCongeSansModificateurs` »

## Questions

Que se passe t'il lorsque l'injection des collaborateurs dans « `DemandeConge` » est faite « `byName` » ?

Que faudrait-il changer pour pouvoir utiliser ce type d'injection?

## Question

Déclarer dans le fichier de configuration Spring un autre collaborateur c2 et utiliser « byType » pour DemandeConge.

Que se passe t'il ?

## Méthode 2 : autowiring avec annotations

Pour activer ce type d'autowiring, il faut ajouter le Bean suivant dans le fichier de configuration Spring :

```
<bean
   class="org.springframework.beans.factory.annotation.
          AutowiredAnnotationBeanPostProcessor"/>
```

Supprimer de la déclaration des Beans toute marque d'injection de collaborateurs et d'autowiring XML.

# TP2 - Injection automatique des collaborateurs

Avec l'annotation `@Autowired`, annoter les classes pour que l'injection soit effectuée automatiquement.

## Question

Déclarer dans le fichier de configuration Spring un autre collaborateur c2 et lancer la classe de test.

Que se passe t'il ?

## Question

Quelle annotation faut il utiliser pour lever l'ambigüité ?

## Activation alternative :

Il est également possible d'utiliser le schéma « `context` » pour configurer l'injection automatique :

```
<context:annotation-config />
```

au lieu du Bean « `AutowiredAnnotationBeanPostProcessor` »

# TP 3

# Détection automatique des composants

## Objectifs

- Annoter les classes pour qu'elles soient détectées en tant que composants Spring

## Activation de la détection automatique

Configurer le fichier « `applicationContext.xml` » de telle sorte que la détection de composants soit effective pour tout composant situé dans « `com.ecoalis` » et ses sous packages.

# TP3 - Détection automatique des composants

Annoter les classes métier, Dao et service de telle sorte que la classe de test TP3 s'exécute correctement.

Utiliser l’annotation la plus adaptée à chaque type de composant (la moins générique).

# TP 4

# Premiers pas avec Spring MVC

## Objectifs

- Comprendre le rôle des résolveurs, des vues physiques et logiques
- Créer un contrôleur avec annotations
- Récupérer les paramètres d'une requête
- Utiliser les Beans du contexte applicatif

## Introduction

L'application Web fournie se lance avec la commande :

```
mvn jetty:run
```

et est accédée à l'URL : http://localhost:8080/

Le « `DispatcherServlet` » traite toutes les requêtes arrivant sur :

`/ddc` (pour Demandes De Congés)

## Introduction

Le « DispatcherServlet » a son propre fichier de contexte :

```
dispatcher-servlet.xml
```

Dans lequel le contrôleur « HomeController » est déclaré pour traiter les requêtes « …/home »

Le contrôleur « `HomeController` » écrit directement dans la « `ServletResponse` ». Modifier la méthode pour qu'il retourne le nom physique complet de la vue : la page JSP « `home.jsp` »

La correspondance entre le nom de vue logique et physique peut être simplifiée grâce à l'utilisation d'un résolveur de vue.

Configurer dans le fichier de contexte du « `DispatcherServlet` » un résolveur basé sur « `InternalResourceViewResolver` » qui renverra la page « `/WEB-INF/mes_pages/home.jsp` » lorsque le nom logique « `home` » sera demandé.

Modifier également le contrôleur pour qu'il retourne ce nom logique.

## Créer un contrôleur avec des annotations

La page d'accueil permet d'obtenir la liste des demandes de congés du collaborateur, sur l'URL « …/`ddc/listerDdc` ».

Créer le contrôleur « `ListerDdcController` » et permettre la gestion de cette URL.

Ne pas oublier de demander à Spring la détection de ce type de contrôleur, dans le fichier de contexte du servlet.

## Récupérer les paramètres d'une requête

La page qui liste les demandes permet d'en obtenir le détail.

Compléter « `DetailDdcController` » pour qu'il gère l'affichage:

- récupérer le paramètre de la requête « `ddcId` » pour pouvoir solliciter le service (mock)
- mettre l'objet renvoyé par le service dans le modèle pour son affichage dans la page
- retourner la vue « `detailDdc` » avec le modèle précédent

## Utiliser les Beans du contexte applicatif (1/2)

Le servlet « `Dispatcher` » dispose de son propre contexte (fichier « `dispatcher-servlet.xml` ») dans lequel sont définis principalement les contrôleurs et les résolveurs.

Les Beans de type service, DAO, ainsi que la configuration relative à la sécurité sont définis dans les fichiers « `applicationContext-xxx.xml` ».

Tous ces Beans peuvent être utilisés par les contrôleurs.

## Utiliser les Beans du contexte applicatif (2/2)

→Ajouter dans le fichier « `web.xml` » le chargement du fichier « `/WEB-INF/applicationContext.xml` » avec la classe d'écouteur de Spring adéquate.

→Injecter par annotation dans « `DetailDdcController` » le service « `DemandeCongeServiceMock2` »

→Utiliser le à la place de l'autre mock pour récupérer la demande de congé « `ddcId` »

# TP 5

# Spring MVC – gestion des formulaires

## Objectifs

- Utiliser l'annotation `@ModelAttribute`
- Créer et mettre en œuvre un validateur

## Introduction

Ce TP reprend la base corrigée du « TP4 – Premiers pas avec Spring MVC ».

Créer le contrôleur « `CreerDdcController` » :

- mappé globalement sur « `/creerDdc` »
- avec une méthode mappée sur « `GET` » qui prend en argument le « `Model` », y injecte la date courante « `now` » et retourne le nom de logique de la vue
- avec une méthode mappée sur « `POST` » qui récupère la soumission du formulaire (`DemandeConge`) et l'affiche sur la sortie standard
- qui peuple les types de demandes « `typeDdc` »
- qui fournit l'attribut du modèle « `demandeConge` », un objet `DemandeConge` vide

## Correction slide suivant

```
@Controller
@RequestMapping("/creerDdc")
public class CreerDdcController {

   @ModelAttribute("typeDdc")
   public TypeDdc[] populateTypeDdc() {
      return TypeDdc.values();
      }

      @ModelAttribute("demandeConge")
      public DemandeConge getDdc() {
            return new DemandeConge();
      }

      @RequestMapping(method = RequestMethod.GET)
      public String setupForm(Model model) {
            model.addAttribute("now", new SimpleDateFormat("dd/mm/yyyy").format(new Date()));
          return "creerDdc";
      }

      @RequestMapping(method = RequestMethod.POST)
      public String processSubmit(@ModelAttribute DemandeConge dc) {
            System.out.println(dc);
          return null;
      }
}
```

mapping global « /creerDdc »

méthode qui peuple les types de demandes grâce à l'annotation @ModelAttribute

fournit l'attribut du modèle « demandeCongé »

mapping « GET » qui injecte la date courante dans le modèle

mapping « POST » qui récupère l'objet soumis par le formulaire

Compléter la page « `creerDdc.jsp` » avec des balises Spring pour que le formulaire puisse poster l'ensemble des données d'une demande de congé :

- informations sur la période (date début et nombre de jours)
- informations sur le collaborateur (prénom et nom)

Vérifier que les informations postées sont bien récupérées par le contrôleur.

- → Créer la classe « `DemandeCongeValidator` », implémentant l'interface « `Validator` ».
- → L'utilitaire « `ValidationUtils` » permet de peupler le conteneur d'erreurs si les champs ne sont pas correctement remplis. Vérifier également que le nombre de jours est > 0.
- → Mettre en œuvre ce validateur dans la méthode gérant le « POST » du contrôleur.
- → Afficher les erreurs dans la vue avec la balise Spring.

# TP 6

# Spring et JPA

# Objectifs

- Utiliser l'approche JpaTemplate
- Utiliser l'approche JpaDaoSupport

## Introduction

Ce TP a pour objectif l'écriture de DAO selon les deux approches JPA proposées par Spring.

La 3ème approche (100% JPA) est donnée en exemple dans `CollaborateurDaoImpl`, où aucune référence à Spring n'est présente. Spring permet néanmoins d'injecter l'`EntityManager` via l'annotation `@PersistenceContext` grâce au Bean `PersistenceAnnotationBeanPostProcessor`.

Ces classes pourront être gérées par un conteneur EJB 3 car elles respectent complètement le standard.

Le contexte Spring est réparti dans les fichiers « `applicationContext.xml` » et « `applicationContext-test.xml` », qui déclarent l'`EntityManager` JPA, le gestionnaire de transactions et le DAO 100% JPA.

Le fichier « `Collaborateur.db.xml` » contient une base de données de test qui sera chargée en mémoire par l'extension de l'une des nombreuses classes fournies par Spring pour les tests unitaires (cf. `SpringDaoTest`).

Le fichier de configuration JPA (`META-INF/persistence-test.xml`) est référencé par l'`EntityManager`.

## Approche JpaTemplate

→ Créer la classe « `CollaborateurDaoTemplateImpl` », implémentant l'interface « `CollaborateurDaoTemplate` ».

→ Déclarer ce DAO dans le fichier de configuration de Spring

→ Par copier/coller, créer la classe « `CollaborateurDaoTemplateTest` », qui utilise « `CollaborateurDaoTemplate` » au lieu de « `CollaborateurDao` ».

## Approche JpaDaoSupport

→ Créer la classe « `CollaborateurDaoSupportImpl` », implémentant l'interface « `CollaborateurDaoSupport` ».

→ Déclarer ce DAO dans le fichier de configuration de Spring

→ Par copier/coller, créer la classe « `CollaborateurDaoSupportTest` », qui utilise « `CollaborateurDaoSupport` » au lieu de « `CollaborateurDao` ».

# TP 7

# Spring et l'AOP

## Objectifs

- Mettre en œuvre l'AOP avec Spring sans et avec annotations

## Introduction

Il faut dans un premier temps ajouter à Eclipse le plugin AJDT:

`http://download.eclipse.org/tools/ajdt/35/update`

La commande « `mvn eclipse:eclipse` » ajoute la nature de projet « AspectJ Project ».

Ne pas activer le weaver à l'ouverture du projet.

## Méthode 1 : configuration AOP en XML

Compléter le fichier de configuration de Spring pour que la méthode « log » de l'aspect « LoggerAspect » intercepte l'appel à toutes les méthodes des implémentations de MonService.

## Méthode 2 : configuration AOP avec annotations

Supprimer les éléments ajoutés lors de la 1ère méthode du fichier de configuration de Spring.

Utiliser les annotations AspectJ pour obtenir le même résultat.

# TP 8

# Spring Security

## Objectifs

- Mettre en œuvre l'authentification
- Sécuriser les vues
- Sécuriser les composants

## Mise en œuvre de l'authentification (1/2)

Configurer le fichier `web.xml` :

- ajouter l'intercepteur de sécurité
- mettre à jour le chargeur de définition des Beans

## Mise en œuvre de l'authentification (2/2)

dans le fichier de configuration de la sécurité :

- déclarer un gestionnaire d'authentification avec deux utilisateurs
  - dany / 456 rôles = `ROLE_DIRECTION` et `ROLE_USER`
  - felix / 123 rôle = `ROLE_USER`
- protéger toutes les vues avec le rôle `ROLE_USER`
- définir le filtre de déconnexion pour qu'il corresponde à l'URL donnée dans la page « `home.jsp` »

## Sécurisation des vues

Ajouter le message « Bienvenue *prénom* ! » dans « `home.jsp` »

Afficher un message uniquement lorsque l'utilisateur connecté possède le rôle `ROLE_DIRECTION`

## Sécurisation des composants

sécuriser la méthode « `getDemandeConge` » du service « `DemandeCongeServiceMock` » pour le rôle `ROLE_DIRECTION`, avec les `jsr-250`.

Créer le contrôleur « `DirectionController` » qui mappe « `/ddc/direction` » et qui appelle la méthode sécurisée. Essayer avec les deux utilisateurs felix et dany pour vérifier la différence d'accès.

## Question

« `DemandeCongeServiceMock` » est maintenant sécurisé. Pourtant, lors de l'affichage du détail d'une demande avec l'utilisateur « felix », aucun refus d'accès n'est constaté. Pourquoi?