Configurer des beans Spring par programmation

Démonstration 5 du module 3

L'objectif de cette démonstration est de configurer un bean spring par programmation

Déroulement

Compléter l'application DemoSpringBeanApplication

Modification de la couche métier

- Mettre en commentaire les annotations @Service, @Profile, @Autowired et @Repository de la couche BLL et DAL.
- Exécuter l'application, les traces consoles sont :

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Parameter 0 of constructor in fr.eni.demo.controller.TrainerController required a bean of type
'fr.eni.demo.bll.TrainerService' that could not be found.


Action:

Consider defining a bean of type 'fr.eni.demo.bll.TrainerService' in your configuration.
```

- Evidemment, le bean trainerService est inconnu pour Spring pour le moment

Classe de configuration et de définition des beans

- Création d'une classe AppConfiguration.
- Elle va définir le bean de la couche d'accès aux données.
- Elle va définir les 2 beans de la couche métier et leur profil.

```
package fr.eni.demo.configuration;

import org.springframework.context.annotation.*;

import fr.eni.demo.bll.*;
import fr.eni.demo.bll.mock.TrainerServiceMock;
import fr.eni.demo.dal.TrainerDAO;
import fr.eni.demo.dal.mock.TrainerDAOMock;

@Configuration
public class AppConfiguration {

	@Bean
	public TrainerDAO getBeanTrainerDAO() {
		return new TrainerDAOMock();
	}

	@Bean
	@Profile("default")
	public TrainerService getBeanTrainerService() {
		return new TrainerServiceImpl(getBeanTrainerDAO());
	}

	@Bean
	@Profile("dev")
	public TrainerService getBeanTrainerServiceMock() {
		return new TrainerServiceMock();
	}
}
```

- Dans le fichier « application.properties », le paramétrage de profil est en commentaires.
- L'exécution de l'application, produit les traces :

```
…
Appel du construteur TrainerController
…
[Trainer [firstName=Anne-Lise, lastName=Baille, email=abaille@campus-eni.fr],
 Trainer [firstName=Stéphane, lastName=Gobin, email=sgobin@campus-eni.fr],
 Trainer [firstName=Julien, lastName=Trillard, email=jtrillard@campus-eni.fr]]
```

- Il y a les 3 formateurs, c'est donc la classe TrainerServiceImpl qui est utilisée.
- Décommenter le paramétrage du profil « `dev` » ; les traces d'exécutions sont :

```
…
Appel du construteur TrainerController
…
[Trainer [firstName=Anne-Lise, lastName=Baille, email=abaille@campus-eni.fr],
 Trainer [firstName=Stéphane, lastName=Gobin, email=sgobin@campus-eni.fr]]
```

- Il ni y a plus que 2 formateurs, c'est la classe TrainerServiceMock qui est utilisée par Spring.

Ainsi, notre classe centralise la partie back de l'application et sa configuration selon les profils.

- Il n'a pas été nécessaire de modifier la classe TrainerController pour manipuler les beans

configurés par notre classe.

- L'annotation @Autowired est capable de récupérer des beans configurés et définis en programmation.
    - C'est pour cela que c'est souvent utilisé pour les bibliothèques externes
    - Sur leurs composants qui par défaut n'ont pas été définis avec @Component,
    - Utilisation @Bean pour pouvoir les injecter plus tard avec @Autowired

> Remarques :
>
> Dans le cas du chargement d'un bean par nommage. Se rappeler que @Bean, génère le nom du bean selon le nom de la méthode annotée. Si besoin, il est possible de préciser un nom par le paramètre value de l'annotation @Bean
>
> Pour préciser le comportement du bean, on peut utiliser les annotation suivantes : @Scope, @Lazy, @DependsOn, @Primary, @Profile
>
> La portée (scope) par défaut est SINGLETON.
>
> L'autowiring par défaut est « par type » .