Injection de dépendance avec Spring

Démonstration 3 du module 3

Les objectifs de cette démonstration :

Connaitre les différents types d'injection offerts par Spring Core :

- Injection par setter
- Injection par constructeur
- Injection par propriété

Savoir utiliser les annotations Spring : @Autowired, @Component, @Service, @Repository et @Controller

Déroulement

Compléter l'application DemoSpringBeanApplication

Création d'une couche métier

- Notre application va se compléter au fil des démonstrations, en représentant une partie du cœur de métier de l'ENI Ecole.
- Dans cette itération, il faut créer un formateur (nom, prénom, email)
- Et une classe métier bouchon (temporaire) qui permet de les gérer :
    - Ajoutant un formateur
    - Retournant tous les formateurs
- Création de la classe Trainer :

```
package fr.eni.demo.bo;

public class Trainer {
	private String firstName;
	private String lastName;
	private String email;

	public Trainer() {
	}

	public Trainer(String firstName, String lastName, String email) {
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
	}

	public String getFirstName() {
		return firstName;
	}
```

```
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	@Override
	public String toString() {
		return "Trainer [firstName=" + firstName + ", lastName=" + lastName + ", email=" + email +
"]";
	}
}
```

- Création de l'interface TrainerService pour qu'il y ait un faible couplage :

```
package fr.eni.demo.bll;

import java.util.List;

import fr.eni.demo.bo.Trainer;

public interface TrainerService {
	void add(String firstName, String lastName, String email);

	List<Trainer> findAll();
}
```

- Création d'une première implémentation TrainerServiceMock, c'est une version temporaire de la classe métier, elle génère des données locales.
    - Pour rendre le code plus lisible, il est conseillé d'utiliser les annotations sémantiques quand cela est possible.
        - @Repository désignera un composant d'accès aux données,
        - @Service un composant métier
        - et @Controller la couche IHM.
    - Dans notre cas, il s'agit d'un service métier, donc utilisation de l'annotation @Service

```
package fr.eni.demo.bll.mock;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

import fr.eni.demo.bll.TrainerService;
import fr.eni.demo.bo.Trainer;
```

```
@Service
public class TrainerServiceMock implements TrainerService {
	// Solution temporaire - gestion d'une liste de formateur locale
	private static List<Trainer> lstTrainers;

	public TrainerServiceMock() {
		lstTrainers = new ArrayList<Trainer>();
		lstTrainers.add(new Trainer("Anne-Lise", "Baille", "abaille@campus-eni.fr"));
		lstTrainers.add(new Trainer("Stéphane", "Gobin", "sgobin@campus-eni.fr"));
	}

	@Override
	public void add(String firstName, String lastName, String email) {
		Trainer t = new Trainer(firstName, lastName, email);
		lstTrainers.add(t);
	}

	@Override
	public List<Trainer> findAll() {
		return lstTrainers;
	}
}
```

- Pour définir le bean, nous utilisons une version sémantique de @Component : @Service.
- Annotation sur un service métier – permet la gestion des transactions sur cette couche.

1. Injection par setter

- Création de la classe TrainerController :

```
package fr.eni.demo.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import fr.eni.demo.bll.TrainerService;
import fr.eni.demo.bo.Trainer;

@Component
public class TrainerController {

	private TrainerService trainerService;

	public void showAllTrainers() {
		List<Trainer> lstTrainers = trainerService.findAll();
		System.out.println(lstTrainers);
	}

	@Autowired
	public void setTrainerService(TrainerService trainerService) {
		System.out.println("Appel de setTrainerService");
		this.trainerService = trainerService;
	}
}
```

- Le composant TrainerController a maintenant une dépendance vers un nouveau bean nommé trainerService .

- Pour demander à Spring l'injection d'une instance dans le paramètre trainerService de la méthode setTrainerService, il faut :
    - Que le bean de type TrainerService soit défini, c'est-à-dire qu'une implémentation de type TrainerService existe et soit annotée par @Component, ou une annotation équivalente.
    - Que l'annotation @Autowired soit positionnée avant le paramètre trainerService (soit juste devant le paramètre, soit devant la méthode set…)

- Modification de la classe d'exécution pour appeler ce nouveau contrôleur :

```
TrainerController trainerCtrler = ctx.getBean(TrainerController.class);
trainerCtrler.showAllTrainers();
```

- Traces de la console :

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

2. Injection par constructeur

- Pour transmettre la dépendance à TrainerController, il est aussi possible de demander l'injection du service dans le constructeur. Pour cela, il suffit de placer l'annotation @Autowired sur le constructeur ou juste devant le paramètre, et que le bean TrainerService soit défini et connu de Spring.
- Cette façon est à utiliser quand la dépendance est obligatoire (nécessairement utilisée).

```
package fr.eni.demo.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import fr.eni.demo.bll.TrainerService;
import fr.eni.demo.bo.Trainer;

@Component
public class TrainerController {

	private TrainerService trainerService;


	@Autowired
	public TrainerController(TrainerService trainerService) {
		System.out.println("Appel du construteur TrainerController");
		this.trainerService = trainerService;
	}



	public void showAllTrainers() {
		List<Trainer> lstTrainers = trainerService.findAll();
		System.out.println(lstTrainers);
	}
}
```

- Traces de la console :

```
…
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]]
```

3. Injection par propriété (à éviter)

- Cette façon de faire n'est plus une bonne pratique aujourd'hui car elle rend difficile l'écriture des tests unitaires.
- Il faut y préférer les injections par setter ou constructeur.

```
package fr.eni.demo.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import fr.eni.demo.bll.TrainerService;
import fr.eni.demo.bo.Trainer;

@Component
public class TrainerController {
	@Autowired
	private TrainerService trainerService;

	public void showAllTrainers() {
		List<Trainer> lstTrainers = trainerService.findAll();
		System.out.println(lstTrainers);
	}
}
```