

# Conflit d'injection de dépendance par type et Résolution

## Démonstration 4 du module 3

Les objectifs de cette démonstration :

- Reconnaître le conflit d'injection de dépendances par type
- Et le résoudre

## Déroulement

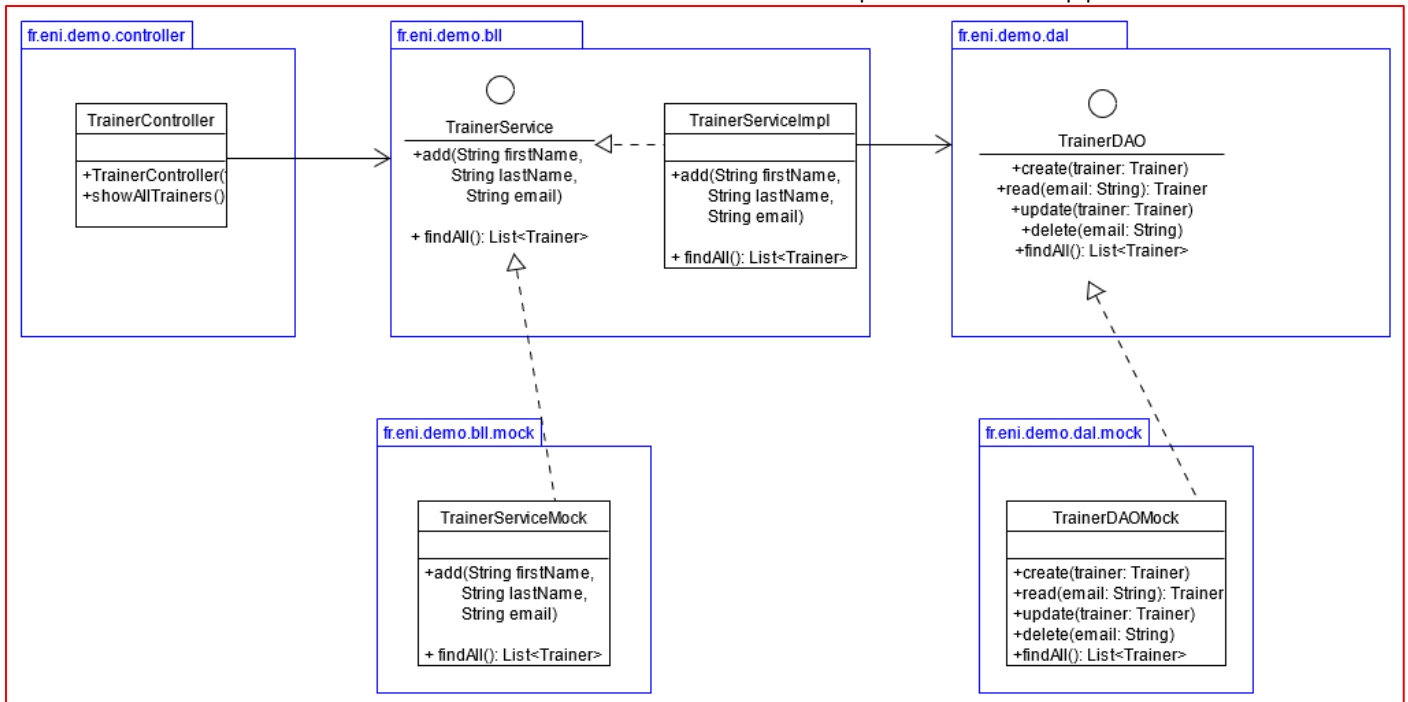
### Compléter l'application DemoSpringBeanApplication

#### Modification de la 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, nous allons commencer à ajouter la classe métier finale pour les formateurs.
- Il y aura donc 2 classes métiers possibles :
  - TrainerServiceImpl
  - TrainerServiceMock

#### Ajout de la classe TrainerServiceImpl

- A cette étape, elle fera appelle à une couche DAL bouchon.



- Création de l'interface TrainerDAO ; elle rassemble les méthodes du CRUD (Create Read Update Delete) et une recherche de tous les formateurs pour le moment.

```
package fr.eni.demo.dal;

import java.util.List;

import fr.eni.demo.bo.Trainer;

public interface TrainerDAO {
    void create(Trainer trainer);

    Trainer read(String email);

    void update(Trainer trainer);

    void delete(String email);

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

- Création de l'implémentation bouchon de cette interface.
  - Elle va manipuler une liste locale comme pour le bouchon de la couche métier
  - Utilisation de l'annotation @Repository ; pour désigner un composant d'accès aux données

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

import java.util.*;

import org.springframework.stereotype.Repository;

import fr.eni.demo.bo.Trainer;
import fr.eni.demo.dal.TrainerDAO;

@Repository
public class TrainerDAOMock implements TrainerDAO {
```

```
// Solution temporaire - gestion d'une liste de formateur locale
private static List<Trainer> LstTrainers;

public TrainerDAOMock() {
    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"));
    // Ajout d'un formateur pour différencier les bouchons des couches DAL et BLL
    LstTrainers.add(new Trainer("Julien", "Trillard", "jtrillard@campus-eni.fr"));
}

@Override
public void create(Trainer trainer) {
    LstTrainers.add(trainer);
}

@Override
public Trainer read(String email) {
    for (Trainer trainer : LstTrainers) {
        if(trainer.getEmail().equals(email)) {
            return trainer;
        }
    }
    return null;
}

@Override
public void update(Trainer trainer) {
    Trainer current = read(trainer.getEmail());
    if(current != null) {
        current.setFirstName(trainer.getFirstName());
        current.setLastName(trainer.getLastName());
    }
}

@Override
public void delete(String email) {
    Iterator<Trainer> it = LstTrainers.iterator();

    while(it.hasNext()) {
        Trainer current = it.next();
        if(current.getEmail().equals(email)) {
            it.remove();
            break;
        }
    }
}

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

- La classe `TrainerServiceImpl` qui implémente l'interface `TrainerService` et utilise la couche DAL ; en s'injectant par type `TrainerDAO`

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

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

- Exécution de l'application après l'ajout du service supplémentaire.
- Traces de la console :

```

2022-03-15 10:22:07.080 INFO 25684 --- [main] fr.eni.demo.DemoSpringBeanApplication : No
Starting DemoSpringBeanApplication using Java 11.0.12 on EC35P1B0C433 with PID 25684
(C:\workspace\220_Spring\demo-spring-bean-application\bin\main started by abaille in
C:\workspace\220_Spring\demo-spring-bean-application)
2022-03-15 10:22:07.082 INFO 25684 --- [main] fr.eni.demo.DemoSpringBeanApplication : No
active profile set, falling back to 1 default profile: "default"
2022-03-15 10:22:07.459 WARN 25684 --- [main] s.c.a.AnnotationConfigApplicationContext :
Exception encountered during context initialization - cancelling refresh attempt:
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name
'trainerController' defined in file [C:\workspace\220_Spring\demo-spring-bean-
application\bin\main\fr\eni\demo\controller\TrainerController.class]: Unsatisfied dependency expressed
through constructor parameter 0; nested exception is
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type
'fr.eni.demo.bll.TrainerService' available: expected single matching bean but found 2:
trainerServiceImpl,trainerServiceMock
2022-03-15 10:22:07.467 INFO 25684 --- [main] ConditionEvaluationReportLoggingListener :

Error starting ApplicationContext. To display the conditions report re-run your application with 'debug'
enabled.
2022-03-15 10:22:07.498 ERROR 25684 --- [main] o.s.b.d.LoggingFailureAnalysisReporter :

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

```



Parameter 0 of constructor in fr.eni.demo.controller.TrainerController required a single bean, but 2 were found:

- trainerServiceImpl: defined in file [C:\workspace\220\_Spring\demo-spring-bean-application\bin\main\fr\eni\demo\bll\TrainerServiceImpl.class]
- trainerServiceMock: defined in file [C:\workspace\220\_Spring\demo-spring-bean-application\bin\main\fr\eni\demo\bll\mock\TrainerServiceMock.class]

#### Action:

Consider marking one of the beans as `@Primary`, updating the consumer to accept multiple beans, or using `@Qualifier` to identify the bean that should be consumed

- L'application est en échec, elle nous précise :
  - Qu'il y a 2 bean qui peuvent être utilisés pour le constructeur de TrainerController :
    - `trainerServiceImpl`
    - `trainerServiceMock`
  - Elle nous propose des actions avec utilisation de :
    - `@Primary`
    - Ou `@Qualifier`

#### 1. Résolution avec `@Qualifier`

- `@Qualifier` permet de faire une injection par nom.
  - Comme les 2 classes ont des noms différents, leur bean associé aussi.
  - En précisant le nom « `trainerServiceImpl` », Spring sera quel bean sélectionner.
- Modification de la classe TrainerController pour mettre en place cette annotation :

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

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
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(@Qualifier("trainerServiceImpl") TrainerService trainerService) {
        System.out.println("Appel du constructeur TrainerController");
        this.trainerService = trainerService;
    }

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

- Traces de la console :

```
...
Appel du constructeur 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]]
```

- L'application est de nouveau opérationnelle. Et Spring injecte le bon service au contrôleur.
- Inconvénient de cette solution, il faut modifier le contrôleur pour sélectionner le bean par nom.

## 2. Résolution avec @Primary

- @Primary : permet à Spring Boot d'utiliser ce bean pour toutes les injections par type associées.
- En déplaçant cette annotation sur les différents beans possibles, il est aisé de lever l'ambiguïté
  - Dans ce cas, placer cette annotation sur la classe TrainerServiceImpl permettra de gérer le conflit.

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

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Service;

import fr.eni.demo.bo.Trainer;
import fr.eni.demo.dal.TrainerDAO;

@Service
@Primary
public class TrainerServiceImpl implements TrainerService {
```

- Il ne sera plus nécessaire de modifier le contrôleur pour charger par nom

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

import java.util.List;

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

import fr.eni.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 constructeur TrainerController");
        this.trainerService = trainerService;
    }
}
```

- L'application est opérationnelle et retourne les même trace qu'avec la solution @Qualifier

### 3. Résolution avec @Profile

#### Définir sur les 2 classes services avec un profil différent

- Sur la classe bouchon @Profile("dev")

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

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

import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Service;

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

@Service
@Profile("dev")
public class TrainerServiceMock implements TrainerService {
```

- Sur la classe TrainerServiceImpl @Profile("default") (retirer l'annotation @Principal)
- "default" : est le profil par défaut pour Spring Boot

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

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Service;

import fr.eni.demo.bo.Trainer;
import fr.eni.demo.dal.TrainerDAO;

@Service
@Profile("default")
public class TrainerServiceImpl implements TrainerService {
```

#### Sélection du profil

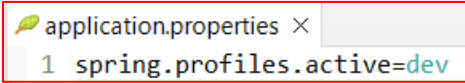
- Si vous exécutez l'application à présent, tout fonctionne et les traces de la console seront :

```
...
Appel du constructeur 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]]
```

- Spring Boot à utiliser le bean déclaré avec le profil "default" ;(TrainerServiceImpl) par défaut pour l'injection de dépendance.
  - Il y a bien les 3 formateurs comme déclaré dans la couche DAL actuelle.

- Sélection du profil "dev"
  - Il faut ajouter dans le fichier de configuration «application.properties» situé dans le dossier «resources» :

```
spring.profiles.active=dev
```



application.properties ×  
1 spring.profiles.active=dev

- Exécution de l'application à présent, les traces de la console sont :

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

- Spring Boot à utiliser le bean déclaré avec le profil "dev" ;(TrainerServiceMock) pour l'injection de dépendance.
  - Il n'y a plus que 2 formateurs comme déclaré dans cette classe bouchon.
- Avantage de cette solution, pouvoir différencier des beans de tests ou temporaires par rapport aux beans de production.
- Il faut penser à mettre en commentaire la ligne suivante dans « application.properties », pour revenir en mode @Profile("default") :

```
#spring.profiles.active=dev
```

- Il sera possible quand nous aurons la version définitive de la couche DAL, de placer le profil développement à la classe bouchon de couche DAL (TrainerDAOMock)
  - Ainsi, ce bean ne sera plus utilisé par Spring pour les injections.