Utilisation de Lombok et BO

# Démonstration 4 du module 1

|  |
| --- |
| Les objectifs de cette démonstration :   * Création de BO avec Lombok * Utilisation du pattern Builder * Création de tests unitaires avec Mockito pour valider les comportements |

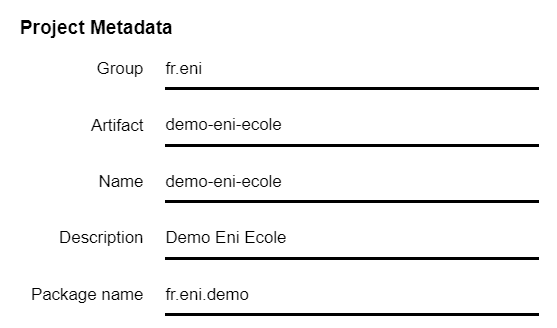
Déroulement

# Contexte

* 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 employé (id, nom, prenom, email, numDom, numPortable, immatriculation)
  + Il est possible de créer un employé complet
  + Il est possible de renseigner seulement un 1 des 2 attributs suivants : numPortable ou numDom

# Créer une application DemoEniEcole avec SpringBoot

* Utiliser Spring initializr et sélectionner :
  + Gradle (privilégier avec le langage Groovy)
  + Java en version minimum 17
  + La version courante de Spring Boot (Release)
  + Le packaging est en Jar
  + Métadonnées :



* + Ajouter les dépendances : Lombok et Developer Tools

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* Importer le projet dans votre IDE, en tant que projet Gradle

# BO avec Lombok

Lombok est une librairie qui permet de ne plus coder les méthodes les plus basiques en Java

* Dans le cadre d’un BO, il va permettre d’omettre la génération des méthodes associées au pattern POJO (Getter/Setter, toString, …)
* Création de la classe Employe :

**package** fr.eni.demo.bo;

**public** **class** Employe {

//Attributs

**private** Integer id;

**private** String nom;

**private** String prenom;

**private** String email;

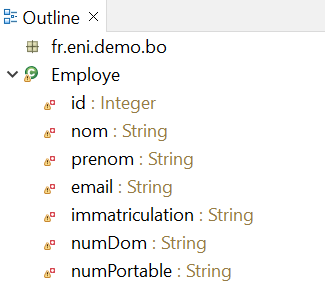
**private** String immatriculation;

**private** String numDom;

**private** String numPortable;

}

* + Regarder Outline, pour le moment, il y a seulement les attributs de la classe



* + Et l’IDE se plaint qu’ils ne sont pas utilisés et visibles
    - Il réclame la création des Getter/Setter
* Ajouter l’annotation @Data

**package** fr.eni.demo.bo;

**import** lombok.Data;

@Data

**public** **class** Employe {

//Attributs

**private** Integer id;

**private** String nom;

**private** String prenom;

**private** String email;

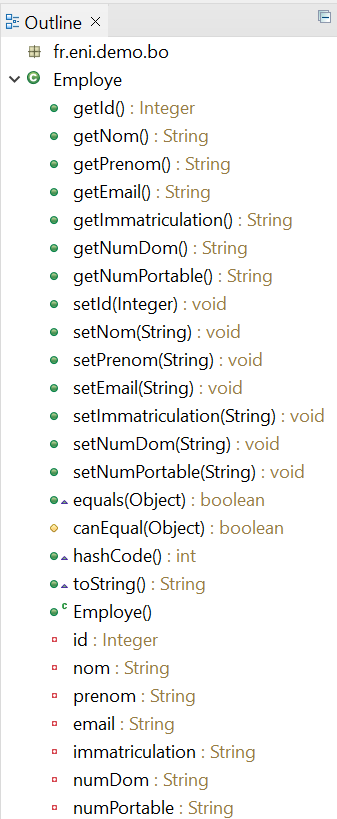
**private** String immatriculation;

**private** String numDom;

**private** String numPortable;

}

* + Regarder Outline



* + Lombok a créé automatiquement tout le design pattern POJO
    - Constructeur sans paramètre
    - Getter/Setter
    - toString
    - equals/hashCode

Si vous êtes dans le cas, où les redéfinitions par défaut vous conviennent

Utilisez @Data

* Cependant, si le métier demande à préciser l’égalité entre 2 instances de votre classe BO ou un toString particulier
  + Exemple égalité uniquement sur immatriculation dans ce métier
* Dans le cas, où il faut détailler les annotations
  + Mettre en commentaire l’annotation @Data
  + Ajouter les annotations pour respecter le design pattern POJO
    - @NoArgsConstructor : génère un constructeur sans argument et public
    - @Getter : pour la génération des getters pour tous les champs
    - @Setter : pour la génération des setters pour tous les champs
    - @EqualsAndHashCode(of= {"immatriculation"}) : génère les méthodes equals et hashCode sur les champs donnés avec of
    - @ToString(of=…) : cette annotation génère la méthode toString (on peut spécifier les champs avec (of=…)).

**package** fr.eni.demo.bo;

**import** lombok.EqualsAndHashCode;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@NoArgsConstructor

@Getter

@Setter

@EqualsAndHashCode(of= {"immatriculation"})

@ToString

//@Data

**public** **class** Employe {

//Attributs

**private** Integer id;

**private** String nom;

**private** String prenom;

**private** String email;

**private** String immatriculation;

**private** String numDom;

**private** String numPortable;

}

* Regarder Outline, les éléments sont les mêmes qu’avec @Data
  + La seule particularité serait si on voulait comparer 2 employés
  + Maintenant, ils sont comparés que par l’immatriculation

# Manipulation du pattern **Builder Pattern**

* Placer l’annotation @Builder sur la classe BO

…

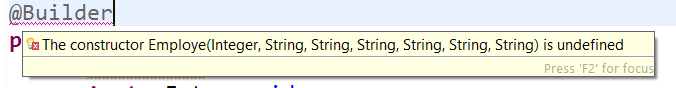
**import** lombok.Builder;

…

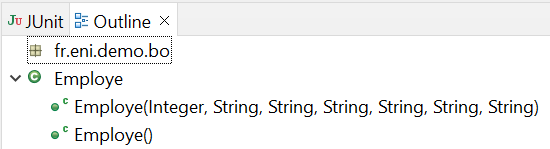
@Builder

**public** **class** Employe {

* L’IDE se plaint qu’il n’y a pas le constructeur avec tous les paramètres



* Ajouter l’annotation @AllArgsConstructor
  + Plus d’erreur au niveau de l’IDE
  + Et dans Outline : 2 constructeurs sont définis (sans paramètre et avec tous les paramètres)



Avec l’annotation @Data, l’annotation @Builder peut ajouter le constructeur avec paramètres.

Cependant le constructeur sans argument disparait.

Il faut donc ajouter les 2 annotations @NoArgsConstructor et @AllArgsConstructor pour que tout soit opérationnel

## Mise en avant du pattern Builder par le test

* Création d’un test unitaire pour voir le comportement du pattern Builder
  + Dans src/test/java ; créer une nouvelle classe de tests TestPatternLombok sous le package fr.eni.demo.bo
  + Utilisation du framework Mockito pour créer les bouchons

@SpringBootTest est une surcouche au test unitaire de JUnit avec l’injection du contexte de Spring automatiquement

* + Lombok permet aussi d’injecter un logger dans les classes avec l'annotation @Slf4j

L’annotation @Slf4j simplifie encore l’utilisation des logs dans le code Java et Spring (Clear Code)

* Créer 2 tests :
  + Le premier avec un employé qui a tous les attributs définis
  + Le deuxième avec un employé qui a que le numéro de son domicile de référencé
  + Ne pas oublier que les tests unitaires sont indépendants.

**package** fr.eni.demo.bo;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.jupiter.api.Assertions.*assertNull*;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.context.SpringBootTest;

**import** lombok.extern.slf4j.Slf4j;

@SpringBootTest

@Slf4j

**public** **class** TestPatternLombok {

@Test

**void** test\_patternBuilder\_TousLesAttributs() {

Employe annelise = Employe

.*builder*()

.id(1)

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0299XXXXXX")

.numPortable("0699XXXXXX")

.build();

***log***.info("test\_patternBuilder\_TousLesAttributs : " + annelise.toString());

*assertThat*(annelise.getNom()).isEqualTo("BAILLE");

*assertThat*(annelise.getImmatriculation()).isEqualTo("ENI\_Ecole\_012892");

*assertThat*(annelise.getNumDom()).isEqualTo("0299XXXXXX");

*assertThat*(annelise.getNumPortable()).isEqualTo("0699XXXXXX");

}

@Test

**void** test\_patternBuilder\_QuelquesAttributs() {

Employe stephane = Employe

.*builder*()

.id(2)

.nom("GOBIN")

.prenom("Stéphane")

.email("sgobin@campus-eni.fr")

.immatriculation("ENI\_Ecole\_011112")

.numDom("0288XXXXXX")

.build();

***log***.info("test\_patternBuilder\_QuelquesAttributs : " + stephane.toString());

*assertThat*(stephane.getNom()).isEqualTo("GOBIN");

*assertThat*(stephane.getImmatriculation()).isEqualTo("ENI\_Ecole\_011112");

*assertThat*(stephane.getNumDom()).isEqualTo("0288XXXXXX");

*assertNull*(stephane.getNumPortable());

}

}

* A l’exécution les tests sont verts
  + Et au niveau des traces nous avons

INFO … --- [ main] fr.eni.demo.bo.TestPatternLombok : test\_patternBuilder\_QuelquesAttributs : Employe(id=2, nom=GOBIN, prenom=Stéphane, email=sgobin@campus-eni.fr, immatriculation=ENI\_Ecole\_011112, numDom=0288XXXXXX, numPortable=null)

INFO … --- [ main] fr.eni.demo.bo.TestPatternLombok : test\_patternBuilder\_TousLesAttributs : Employe(id=1, nom=BAILLE, prenom=Anne-Lise, email=abaille@campus-eni.fr, immatriculation=ENI\_Ecole\_012892, numDom=0299XXXXXX, numPortable=0699XXXXXX)