# Map Struct

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

# Plan


1. Introduction
2. Préparation du projet
3. Solution sans `Map Struct`
4. Solution avec `Map Struct`
   - Configuration de `pom.xml`
   - Extension Eclipse
   - `@Mapper`
   - `@Mapping`
   - `@AfterMapping`
   - `@BeforeMapping`
   - Mapper une collection

# Spring Boot

## **DTO** : **D**ata **T**ransfer **O**bject

- Design pattern
- Souvent utilisés en conjonction des objets d'accès aux données (entités ou modèles).
- À ne pas confondre avec le **DAO** (**D**ata **A**ccess **O**bject) et le **DAL** (**D**ata **A**ccess **L**ayer)

# Spring Boot

## Map Struct

- Mapper : générateur de code basé sur les annotations
- Convertisseur d'objets **Java** : **DTO** (**D**ata **T**ransfer **O**bject) en entités ou l'inverse
- Plugin qui complète le compilateur **Java**
- Documentation officielle : `https://mapstruct.org/documentation/stable/reference/html/`

# Spring Boot

## Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `spring-boot-map-struct` dans `Name`,
  - `com.example` dans `Group`,
  - `spring-boot-map-struct` dans `Artifact`
  - `com.example.demo` dans `Package`
- Cliquer sur `Next`
- Chercher et cocher les cases correspondantes aux `Spring Data JPA`, `MySQL Driver`, `Lombok`, `Spring Web` et `Spring Boot DevTools`
- Cliquer sur `Next` puis sur `Finish`

# Spring Boot

## Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

# Spring Boot

## Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

## Pour la suite, nous considérons

- une entité `Personne` à définir dans `com.example.demo.model`
- une interface DAO `PersonneRepository` à définir dans `com.example.demo.dao`
- un contrôleur REST `PersonneController` à définir dans `com.example.demo.controller`

# Spring Boot & REST

**Créons une entité** `Personne` **dans** `com.example.demo.model`

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Personne {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long num;
        @NonNull
        private String nom;
        @NonNull
        private String prenom;
        @NonNull
        @Column(unique = true)
        private String nomUtilisateur;
        @NonNull
        private String motDePasse;
        @NonNull
        private LocalDate dateInscription;
        @NonNull
        private Character genre;
}
```

# Spring Boot & REST

**Préparons notre interface DAO** `PersonneRepository`

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.model.Personne;

public interface PersonneRepository extends JpaRepository<Personne,
   Long> {

}
```

# Spring Boot

**Créons une interface** `PersonneService` **dans** `com.example.demo.service`

```
package com.example.demo.service;

import java.util.Collection;

import com.example.demo.model.Personne;

public interface PersonneService {

        public Collection<Personne> findAll();

        public Personne findById(Long id);

        public Personne save(Personne personne);

}
```

**Dans ce même package, implémentons l'interface précédente**

```
package com.example.demo.service;

import java.util.Collection;

import org.springframework.stereotype.Service;

import com.example.demo.dao.PersonneRepository;
import com.example.demo.model.Personne;

import lombok.AllArgsConstructor;

@Service
@AllArgsConstructor
public class PersonneServiceImpl implements PersonneService {

        private PersonneRepository personneRepository;

        public Collection<Personne> findAll() {
                return personneRepository.findAll();
        }

        public Personne findById(Long id) {
                return personneRepository.findById(id).orElse(null);
        }


        public Personne save(Personne personne) {
                return personneRepository.save(personne);
        }
}
```

**Créons le contrôleur REST suivant dans** `com.example.demo.controller`

```
@RestController
@AllArgsConstructor
public class PersonneRestController {

        private PersonneService personneService;

        @GetMapping("/personnes")
        public Collection<Personne> getPersonnes() {
                return personneService.findAll();
        }

        @GetMapping("/personnes/{id}")
        public Personne getPersonne(@PathVariable("id") long id) {
                return personneService.findById(id);
        }

        @PostMapping("/personnes")
        public Personne addPersonne(@RequestBody Personne personne) {
                return personneService.save(personne);
        }
}
```

# Spring Boot

**Dans `application.properties`, ajoutons les données permettant la connexion à la base de données et la configuration de `Hibernate`**

```
server.servlet.context-path = /ws
spring.datasource.url = jdbc:mysql://localhost:3306/map_struct?createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.
  PhysicalNamingStrategyStandardImpl
```



L'ajout de la propriété `spring.jpa.hibernate.naming.physical-strategy` permet de forcer **Hibernate** à utiliser les mêmes noms pour les tables et les colonnes que les entités et les attributs.

# Spring Boot

**Pour alimenter la base de données avec quelques données au démarrage de l'application**

```
@SpringBootApplication
public class SpringBootMapStructApplication {

        public static void main(String[] args) {
                SpringApplication.run(SpringBootMapStructApplication.class, args);
        }

    @Bean
    CommandLineRunner start(PersonneRepository personneRepository) {
        return args -> {
            personneRepository.save(new Personne("wick", "john", "wick", "wick", LocalDate.of
               (2023, 1, 1), 'H'));
            personneRepository.save(new Personne("dalton", "jack", "jack", "jack", LocalDate.of
               (2023, 1, 1), 'H'));

        };
    }
}
```

# Spring Boot

**Pour tester, allez à `http://localhost:8080/ws/personnes` et vérifiez l'affichage suivant**


```
[
    {
        "num": 1,
        "nom": "wick",
        "prenom": "john",
        "nomUtilisateur": "wick",
        "motDePasse": "wick",
        "dateInscription": "2023-01-01",
        "genre": "H"
    },
    {
        "num": 2,
        "nom": "dalton",
        "prenom": "jack",
        "nomUtilisateur": "jack",
        "motDePasse": "jack",
        "dateInscription": "2023-01-01",
        "genre": "H"
    }
]
```

# Spring Boot

## Autour de la classe `Personne`

- `Personne` : correspond à une table de la base de données
- `PersonneRequestDto` : correspond au modèle de données envoyées par le client
- `PersonneResponseDto` : correspond au modèle de données à retourner au client

# Spring Boot

**Contenu de** `Personne`

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Personne {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long num;
        @NonNull
        private String nom;
        @NonNull
        private String prenom;
        @NonNull
        @Column(unique = true)
        private String nomUtilisateur;
        @NonNull
        private String motDePasse;
        @NonNull
        private LocalDate dateInscription;
        @NonNull
        private Character genre;
}
```

# Spring Boot

**Contenu de** `PersonneRequestDto`

```
@Data
@Builder
public class PersonneRequestDto {
    private Long num;
    private String lastname;
    private String firstname;
    private String username;
    private String password;
    private String gender;
}
```

**Contenu de** `PersonneResponseDto`

```
@Data
@Builder
public class PersonneResponseDto {
    private Long num;
    private String lastname;
    private String firstname;
    private String username;
    private LocalDate
       registrationDate;
    private String gender;
}
```

# Spring Boot

**Commençons par modifier l'interface de** `PersonneService`

```
package com.example.demo.service;

import java.util.Collection;

import com.example.demo.dto.PersonneRequestDto;
import com.example.demo.dto.PersonneResponseDto;

public interface PersonneService {

        public Collection<PersonneResponseDto> findAll();

        public PersonneResponseDto findById(Long id);

        public PersonneResponseDto save(PersonneRequestDto personne);

}
```

# Spring Boot

**Commençons par définir la méthode `findAll` dans `PersonneServiceImpl`**

```
@Service
@AllArgsConstructor
public class PersonneServiceImpl implements PersonneService {
    private PersonneRepository personneRepository;

    public Collection<PersonneResponseDto> findAll() {
        var personnes = personneRepository.findAll();
        List<PersonneResponseDto> resultat = new ArrayList<>();
        for (var objet : personnes) {
            var personneDto = new PersonneResponseDto();
            personneDto.setNum(objet.getNum());
            personneDto.setLastname(objet.getNom());
            personneDto.setFirstname(objet.getPrenom());
            personneDto.setGender(objet.getGenre() + "");
            personneDto.setRegistrationDate(objet.getDateInscription());
            personneDto.setUsername(objet.getNomUtilisateur());
            resultat.add(personneDto);
        }
        return resultat;
    }

    public PersonneResponseDto findById(Long id) {
        return null;
    }

    public PersonneResponseDto save(PersonneRequestDto personne) {
        return null
    }
}
```

# Spring Boot

**Nouvelle version du contrôleur**

```
@RestController
@AllArgsConstructor
public class PersonneRestController {

    private PersonneService personneService;

    @GetMapping("/personnes")
    public Collection<PersonneResponseDto> getPersonnes() {
        return personneService.findAll();
    }

    @GetMapping("/personnes/{id}")
    public PersonneResponseDto getPersonne(@PathVariable long id) {
        return personneService.findById(id);
    }

    @PostMapping("/personnes")
    public PersonneResponseDto addPersonne(@RequestBody PersonneRequestDto personne) {
        return personneService.save(personne);
    }
}
```

# Spring Boot

**Pour tester, allez à `http://localhost:8080/ws/personnes` et vérifiez l'affichage suivant**


```
[
    {
        "num": 1,
        "lastname": "wick",
        "firstname": "john",
        "username": "wick",
        "registrationDate": "1970-01-01",
        "gender": "H"
    },
    {
        "num": 2,
        "lastname": "dalton",
        "firstname": "jack",
        "username": "jack",
        "registrationDate": "1970-01-01",
        "gender": "H"
    }
]
```

# Spring Boot

## Constat

Code précédent long

# Spring Boot

## Constat

Code précédent long

## Solution

Utiliser **Map Struct**

# Spring Boot & REST

**Dans `properties` de `pom.xml`, ajoutons la balise suivante**

```
<org.mapstruct.version>1.6.3</org.mapstruct.version>
```

# Spring Boot & REST

**Dans** `properties` **de** `pom.xml`**, ajoutons la balise suivante**

```
<org.mapstruct.version>1.6.3</org.mapstruct.version>
```

**Dans** `dependencies` **de** `pom.xml`**, ajoutons la dépendance suivante**

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${org.mapstruct.version}</version>
</dependency>
```

# Spring Boot & REST

**Dans `plugins` de `pom.xml`, ajoutons le plugin suivant**

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
        <source>17</source> <!-- depending on your project -->
        <target>17</target> <!-- depending on your project -->
        <annotationProcessorPaths>
            <path>
                <groupId>org.mapstruct</groupId>
                <artifactId>mapstruct-processor</artifactId>
                <version>${org.mapstruct.version}</version>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

# Spring Boot

## Extension **Eclipse** pour faciliter le mapping

**MapStruct Eclipse Mapping**

# Spring Boot

**Commençons par une interface de mapping appelée `PersonneMapper` et ajoutons l'attribut `componentModel = "spring"` à `@Mapper` pour permettre l'injection Spring**

```
package com.example.demo.mapper;

@Mapper(componentModel = "spring")
public interface PersonneMapper {

}
```

# Spring Boot

**Définissons les deux méthodes de mapping**

```
package com.example.demo.mapper;

@Mapper(componentModel = "spring")
public interface PersonneMapper {

    PersonneResponseDto fromPersonneToPersonneResponseDto(Personne personne);

    Personne fromPersonneRequestDtoToPersonne(PersonneRequestDto personneRequestDto);

}
```

# Spring Boot

**Ajoutons l'annotation `@Mapping` permettant de mapper les attributs de deux classes**

```
@Mapper(componentModel = "spring")
public interface PersonneMapper {

        @Mapping(source = "dateInscription", target = "registrationDate")
        @Mapping(source = "nom", target = "lastname")
        @Mapping(source = "prenom", target = "firstname")
        @Mapping(source = "nomUtilisateur", target = "username")
        @Mapping(source = "genre", target = "gender")
        PersonneResponseDto fromPersonneToPersonneResponseDto(Personne personne);

        @Mapping(target = "dateInscription", ignore = true)
        @Mapping(source = "password", target = "motDePasse")
        @Mapping(source = "lastname", target = "nom")
        @Mapping(source = "firstname", target = "prenom")
        @Mapping(source = "username", target = "nomUtilisateur")
        @Mapping(source = "gender", target = "genre")
        Personne fromPersonneRequestDtoToPersonne(PersonneRequestDto personneRequestDto);
}
```

# Spring Boot

## Explication

- `source` : indique le nom de l'attribut dans la classe d'entrée (le paramètre)
- `target` : indique le nom de l'attribut dans la classe de sortie (le type de retour)
- `ignore` : indique que l'attribut n'est pas requis pour le mapping

# Spring Boot

**Injectons le mapper dans l'interface de** `PersonneServiceImpl`

```
@Service
@AllArgsConstructor
public class PersonneServiceImpl implements PersonneService {

        private PersonneRepository personneRepository;
        private PersonneMapper mapper;

}
```

# Spring Boot

**Utilisons le mapper dans les trois méthodes précédentes du service**

```
public Collection<PersonneResponseDto> findAll() {
    var personnes = personneRepository.findAll().stream()
                    .map(elt -> mapper.fromPersonneToPersonneResponseDto(elt))
                    .collect(Collectors.toList());
    return personnes;
}

public PersonneResponseDto findById(Long id) {
    return mapper.fromPersonneToPersonneResponseDto(personneRepository.findById(id)
    .orElse(null));
}

public PersonneResponseDto save(PersonneRequestDto personne) {
    var p = mapper.fromPersonneRequestDtoToPersonne(personne);
    p.setDateInscription(LocalDate.now());
    var result = personneRepository.save(p);
    return mapper.fromPersonneToPersonneResponseDto(result);
}
```

# Spring Boot

**Objet envoyé avec** `POST`

```
{
    "lastname": "casanova",
    "password": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "gender": "F"
}
```

**Objet retourné par** `POST`

```
{
    "num": 3,
    "lastname": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "registrationDate": "2023-02-07",
    "gender": "F"
}
```

# Spring Boot

**Nous pouvons aussi utiliser la propriété `expression` pour initialiser l'attribut `dateInscription`**

```
@Mapper(componentModel = "spring")
public interface PersonneMapper {

        @Mapping(source = "dateInscription", target = "registrationDate")
        @Mapping(source = "nom", target = "lastname")
        @Mapping(source = "prenom", target = "firstname")
        @Mapping(source = "nomUtilisateur", target = "username")
        @Mapping(source = "genre", target = "gender")
        PersonneResponseDto fromPersonneToPersonneResponseDto(Personne personne);

        @Mapping(target = "dateInscription", expression = "java(java.time.LocalDate.now())")
        @Mapping(source = "password", target = "motDePasse")
        @Mapping(source = "lastname", target = "nom")
        @Mapping(source = "firstname", target = "prenom")
        @Mapping(source = "username", target = "nomUtilisateur")
        @Mapping(source = "gender", target = "genre")
        Personne fromPersonneRequestDtoToPersonne(PersonneRequestDto personneRequestDto);
}
```

# Spring Boot

**Plus besoin d'initialiser** `dateInscription` **dans** `save` **de** `PersonneServiceImpl`

```
public Collection<PersonneResponseDto> findAll() {
    var personnes = personneRepository.findAll().stream()
                    .map(elt -> mapper.fromPersonneToPersonneResponseDto(elt))
                    .collect(Collectors.toList());
    return personnes;
}

public PersonneResponseDto findById(Long id) {
    return mapper.fromPersonneToPersonneResponseDto(personneRepository.findById(id)
    .orElse(null));
}

public PersonneResponseDto save(PersonneRequestDto personne) {
    var p = mapper.fromPersonneRequestDtoToPersonne(personne);
 //   p.setDateInscription(LocalDate.now());
    var result = personneRepository.save(p);
    return mapper.fromPersonneToPersonneResponseDto(result);
}
```

# Spring Boot

**Objet envoyé avec** `POST`

```
{
    "lastname": "casanova",
    "password": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "gender": "F"
}
```

**Objet retourné par** `POST`

```
{
    "num": 3,
    "lastname": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "registrationDate": "2023-02-07",
    "gender": "F"
}
```

# Spring Boot

## Question

Et si nous souhaitons retourner au client une valeur de type `String` (comme `Homme` ou `Femme` pour le `genre`

# Spring Boot

## Question

Et si nous souhaitons retourner au client une valeur de type `String` (comme `Homme` ou `Femme` pour le `genre`

## Solution

Utiliser `@AfterMapping` pour remplacer la valeur mappée par `Homme` ou `Femme`

# Spring Boot

**Ajoutons la méthode** `afterMapping` **dans** `PersonneMapper`

```
@Mapper(componentModel = "spring")
public interface PersonneMapper {
        @Mapping(source = "dateInscription", target = "registrationDate")
        @Mapping(source = "nom", target = "lastname")
        @Mapping(source = "prenom", target = "firstname")
        @Mapping(source = "nomUtilisateur", target = "username")
        @Mapping(source = "genre", target = "gender")
        PersonneResponseDto fromPersonneToPersonneResponseDto(Personne personne);

        @Mapping(target = "dateInscription", expression = "java(java.time.LocalDate.now())")
        @Mapping(source = "password", target = "motDePasse")
        @Mapping(source = "lastname", target = "nom")
        @Mapping(source = "firstname", target = "prenom")
        @Mapping(source = "username", target = "nomUtilisateur")
        @Mapping(source = "gender", target = "genre")
        Personne fromPersonneRequestDtoToPersonne(PersonneRequestDto personneRequestDto);

        @AfterMapping
        default void afterMapping(@MappingTarget PersonneResponseDto dto, Personne personne) {
                if (personne.getGenre() == 'H') {
                        dto.setGender("Homme");
                } else {
                        dto.setGender("Femme");
                }
        }
}
```

# Spring Boot

**Objet envoyé avec** `POST`

```
{
    "lastname": "casanova",
    "password": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "gender": "F"
}
```

**Objet retourné par** `POST`

```
{
    "num": 3,
    "lastname": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "registrationDate": "2023-02-07",
    "gender": "Femme"
}
```

# Spring Boot

## Question

Et si nous souhaitons stocker `H` ou `F` comme genre quelle que soit la valeur envoyée par le client

# Spring Boot

## Question

Et si nous souhaitons stocker `H` ou `F` comme genre quelle que soit la valeur envoyée par le client

## Solution

Utiliser `@BeforMapping` pour modifier la valeur reçue avant de faire le mapping

# Spring Boot

**Ajoutons la méthode** `beforeMapping` **dans** `PersonneMapper`

```
@Mapper(componentModel = "spring")
public interface PersonneMapper {

  // les deux méthodes de mapping précédentes

  @AfterMapping
  default void afterMapping(@MappingTarget PersonneResponseDto dto, Personne personne) {
    if (personne.getGenre() == 'H') {
      dto.setGender("Homme");
    } else {
      dto.setGender("Femme");
    }
  }
  @BeforeMapping
  default void beforeMapping(@MappingTarget Personne personne, PersonneRequestDto dto) {
    if (dto.getGender().equalsIgnoreCase("homme") ||
        dto.getGender().equalsIgnoreCase("masculin") ||
        dto.getGender().equalsIgnoreCase("male") ||
        dto.getGender().equalsIgnoreCase("h")) {
      personne.setGenre('H');
    } else {
      personne.setGenre('F');
    }
  }
}
```

# Spring Boot

**Objet envoyé avec** `POST`

```
{
    "lastname": "casanova",
    "password": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "gender": "Femelle"
}
```

**Objet retourné par** `POST`

```
{
    "num": 3,
    "lastname": "casanova",
    "firstname": "danielle",
    "username": "danielle",
    "registrationDate": "2023-02-07",
    "gender": "Femme"
}
```

# Spring Boot

## Remarque

La méthode `fromPersonneToPersonneResponseDto` permet de mapper un objet (pas une collection)

# Spring Boot

## Remarque

La méthode `fromPersonneToPersonneResponseDto` permet de mapper un objet (pas une collection)

## Mais

Nous pouvons définir une méthode qui permet de mapper une collection.

# Spring Boot

**Ajoutons la méthode qui permet de mapper une liste**

```
@Mapper(componentModel = "spring")
public interface PersonneMapper {
        @Mapping(source = "dateInscription", target = "registrationDate")
        @Mapping(source = "nom", target = "lastname")
        @Mapping(source = "prenom", target = "firstname")
        @Mapping(source = "nomUtilisateur", target = "username")
        @Mapping(source = "genre", target = "gender")
        PersonneResponseDto fromPersonneToPersonneResponseDto(Personne personne);

        @Mapping(target = "dateInscription", expression = "java(java.time.LocalDate.now())")
        @Mapping(source = "password", target = "motDePasse")
        @Mapping(source = "lastname", target = "nom")
        @Mapping(source = "firstname", target = "prenom")
        @Mapping(source = "username", target = "nomUtilisateur")
        @Mapping(source = "gender", target = "genre")
        Personne fromPersonneRequestDtoToPersonne(PersonneRequestDto personneRequestDto);

        List<PersonneResponseDto> fromPersonneToPersonneResponseDto(List<Personne> personnes);

        // + les méthodes précédentes
}
```

# Spring Boot

**Simplifions la méthode `findAll` de `PersonneServiceImpl`**

```
@Service
@AllArgsConstructor
public class PersonneServiceImpl implements PersonneService {
    private PersonneRepository personneRepository;
    private PersonneMapper mapper;

    public Collection<PersonneResponseDto> findAll() {
        return mapper.fromPersonneToPersonneResponseDto(personneRepository.findAll());
    }

    public PersonneResponseDto findById(Long id) {
        return mapper.fromPersonneToPersonneResponseDto(personneRepository.findById(id)
        .orElse(null));
    }

    public PersonneResponseDto save(PersonneRequestDto personne) {
        var p = mapper.fromPersonneRequestDtoToPersonne(personne);
        var result = personneRepository.save(p);
        return mapper.fromPersonneToPersonneResponseDto(result);
    }
}
```