* MapStrcut face conversii dintre diferite obiecte
* Trebuie doar sa cream interfata si implementarea ei se va crea automat la compiletime

**Creare**

1. Ne trebuie dependenta:

<dependency>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct</artifactId>  
 <version>1.5.3.Final</version>  
</dependency>

Atentie! MapStruct nu merge cu ultimele versiuni de LomBok, iata de ce, mai trebuie o dependenta si un path

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok-mapstruct-binding</artifactId>  
 <version>0.2.0</version>  
</dependency>

si asta

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.1</version>  
 <configuration>  
 <source>${java.version}</source>  
 <target>${java.version}</target>  
 <annotationProcessorPaths>  
 <path>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct-processor</artifactId>  
 <version>1.4.2.Final</version>  
 </path>  
 <path>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.28</version>  
 </path>

<path>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok-mapstruct-binding</artifactId>  
 <version>0.2.0</version>  
</path>

</annotationProcessorPaths>  
 </configuration>  
</plugin>

**Atentie la lombok-mapstruct-binding path!!!**

1. Cream User si UserDTO

@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class User {  
 private String username;  
 private String password;  
 private String email;  
 private int age;  
}

@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class UserDTO {  
 private String username;  
 private String password;  
 private String email;  
 private int age;  
  
}

1. Cream interfata de mapat:

@Mapper(componentModel = "spring")  
public interface UserMapper {  
 UserDTO userToUserDTO(User user);  
 User userDTOToUser(UserDTO userDTO);  
}

Numele la metode nu conteaza. Conteaza tipul la parametru si la return, si el auomat va apela getters si setters ale obiectelor.

**componentModel** = “spring” va face sa se genereze un bean de tip UserMapper

@RestController  
public class MyRestController {  
 @Autowired  
 private UserMapper userMapper;  
 @GetMapping  
 public UserDTO get(){  
 User user = User.*builder*()  
 .username("user1")  
 .password("test123")  
 .age(20)  
 .email("test@email.ru")  
 .build();  
  
 return userMapper.userToUserDTO(user);  
 }  
}

**Fara @Autowired**

Putem obtine o implementare la interfata si asa:

private UserMapper userMapper = Mappers.*getMapper*(UserMapper.class);

**Beans in clasa de mapat**

* Daca avem nevoie de beanuri cand mapam, problema e ca nu vom mai putea folosi in interfata
* Vom folosi o clasa bastracta in schimb:

@Mapper(componentModel = "spring")

**public** **abstract** **class** **SimpleDestinationMapperUsingInjectedService** { @Autowired **protected** SimpleService simpleService;

@Mapping(target="name",expression"java(simpleService.enrichName(source.getName()))")

**public** **abstract** SimpleDestination **sourceToDestination**(SimpleSource source);

}

* Asa el va putea folosi membrii din clasa.
* Atentie! Nu punem fieldurile ca private, ca de altfel nu le va putea accesa in subclasa
* expression(“java()”) – permite sa executam o metoda cand mapam un field

**@Mapping**

* Apar probleme daca numele la fielduri difera la obiectele de mapat
* @Getter  
  @Setter  
  @ToString  
  @NoArgsConstructor  
  @AllArgsConstructor  
  @Builder  
  public class User {  
   private String username;  
   private String password;  
   private String email;  
   private int age;  
  }

@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class UserDTO {  
 private String login;  
 private String pass;  
 private String mail;  
 private int age;  
  
}

si maparea se va face asa:

@Mapper(componentModel = "spring")  
public interface UserMapper {  
 @Mapping(target = "login",source = "username")  
 @Mapping(target = "pass",source = "password")  
 @Mapping(target = "mail",source = "email")  
 UserDTO userToUserDTO(User user);  
 @Mapping(target = "username",source = "login")  
 @Mapping(target = "password",source = "pass")  
 @Mapping(target = "email",source = "mail")  
 User userDTOToUser(UserDTO userDTO);  
}

target este mereu obiectul returnat!

**Convert field object**

* User si UserDTO ar putea avea obiecte in ele ce tot ar trebui convertite.
* De ex, fie ca mai cream un Date si DateDTO class:
* @Getter  
  @Setter  
  @NoArgsConstructor  
  @ToString

@Builder

@AllArgsConstructor

public class Date {  
 private int day;  
 private int month;  
 private int year;  
}

@Getter  
@Setter  
@NoArgsConstructor  
@ToString

@Builder

@AllArgsConstructor  
public class DateDTO {  
 private int day;  
 private int month;  
 private int year;  
}

si User si UserDTO:

@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class User {  
 private String username;  
 private String password;  
 private String email;  
 private int age;  
 private Date date;  
}

@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class UserDTO {  
 private String login;  
 private String pass;  
 private String mail;  
 private int age;  
 private DateDTO date;  
  
}

* Trebuie sa scriem metode si pentru a covnerti Date in DateDTO si viceversa, si deja MapStruct va cauta automat metodele de conversie pentru Date si DateDTO
* @RestController  
  public class MyRestController {  
   @Autowired  
   private UserMapper userMapper;  
   @GetMapping  
   public UserDTO get(){  
   User user = User.*builder*()  
   .username("user1")  
   .password("test123")  
   .age(20)  
   .email("test@email.ru")  
   .date(  
   Date.*builder*()  
   .day(21)  
   .month(11)  
   .year(2023)  
   .build()  
   )  
   .build();  
    
   return userMapper.userToUserDTO(user);  
   }  
  }

@Mapper(componentModel = "spring")  
public interface UserMapper {  
 @Mapping(target = "login",source = "username")  
 @Mapping(target = "pass",source = "password")  
 @Mapping(target = "mail",source = "email")  
 UserDTO userToUserDTO(User user);  
 @Mapping(target = "username",source = "login")  
 @Mapping(target = "password",source = "pass")  
 @Mapping(target = "email",source = "mail")  
 User userDTOToUser(UserDTO userDTO);  
  
 DateDTO dateToDateDTO(Date date);  
 Date dateDTOToDate(DateDTO dateDTO);  
}

**Mapping with TypeConversion**

* MapStruct suportt si unele conversii, cum ar fi la Date:

**public** **class** **Employee** {

// other fields

**private** Date startDt;

// getters and

setters

}

**public** **class** **EmployeeDTO**

{ // other fields

**private** String employeeStartDt;

// getters and setters

}

@Mapping(target="employeeId",source="entity.id") @Mapping(target="employeeName",source="entity.name") @Mapping(target="employeeStartDt", source = "entity.startDt", dateFormat = "dd-MM-yyyy HH:mm:ss")

EmployeeDTO **employeeToEmployeeDTO**(Employee entity);

@Mapping(target="id", source="dto.employeeId")

@Mapping(target="name",source="dto.employeeName") @Mapping(target="startDt",source="dto.employeeStartDt", dateFormat="dd-MM-yyyy HH:mm:ss")

Employee **employeeDTOtoEmployee**(EmployeeDTO dto);

**Vedem ca avem dateFormat argument in anotatie.**

**Custom method**

* Putem crea propria metoda de mapare
* Pentru asta, cream o clasa abstracta si oferim un corp metodei:
* @Mapper(componentModel = "spring")  
  public abstract class UserMapper {  
    
   public UserDTO userToUserDTO(User user){  
   return UserDTO.*builder*()  
   .pass("\*\*\*\*\*\*\*\*")  
   .login(user.getUsername())  
   .age(user.getAge())  
   .mail(user.getEmail())  
   .date(dateToDateDTO(user.getDate()))  
   .build();  
   }  
   @Mapping(target = "username",source = "login")  
   @Mapping(target = "password",source = "pass")  
   @Mapping(target = "email",source = "mail")  
   public abstract User userDTOToUser(UserDTO userDTO);  
    
   public abstract DateDTO dateToDateDTO(Date date);  
   public abstract Date dateDTOToDate(DateDTO dateDTO);  
  }

**@BeforeMapping, @AfterMapping si @MappingTarget**