1. **ModelMapper -мапване между 2 обекта**

* Използва се за мапване от Entity -> DTO или DTO -> Entity
* DTO – Data Transfer Object

-dependency

<dependency>  
 <groupId>org.modelmapper</groupId>  
 <artifactId>modelmapper</artifactId>  
 <version>3.1.1</version>  
</dependency>

* **SKIP() – пропуска мапване**

typeMap.addMappings(mapper -> mapper.skip(Employee::setSalary)); typeMap.map(employeeDto);

* **MAPPING – with same field names and same types**
* modelMapper.map(employee, employeeDto);
* **MAPPING – typeMap – with a different field name, but same type**
* Създаване на нов typeMap

ModelMapper modelMapper = new ModelMapper();

TypeMap typeMap = mapper.createTypeMap( EmployeeDto.class, Employee.class);

typeMap.addMappings(m -> m.map(src -> src.getName(), Employee::setFirtsName));

typeMap.map(employeeDto);

* Използване на .typeMap(“source class”, “destination class”).

.addMapping(“source getter”, “destination setter) –

* Converter-a се пише преди .typeMap

@Bean  
public ModelMapper modelMapper() {  
 ModelMapper modelMapper = new ModelMapper();  
  
 modelMapper  
 .typeMap(GameAddDto.class, Game.class)  
 .addMapping(GameAddDto::getImageThumbnail, Game::setImageThumbnail);

Converter<String, LocalDate> localdateConverter = new Converter<String, LocalDate>() {  
 @Override  
 public LocalDate convert(MappingContext<String, LocalDate> mappingContext) {  
 return mappingContext == null  
 ? null  
 : LocalDate.*parse*(mappingContext.getSource(), DateTimeFormatter.*ofPattern*("dd-MM-yyyy"));  
 }  
 };  
 modelMapper.addConverter(localdateConverter);  
  
 return modelMapper;

* **CONVERTER – with a different type of fields**

ModelMapper modelMapper = new ModelMapper();

Converter toUppercase = ctx -> ctx.getSource() == null ? null : ctx.getSource().toUpperCase();

TypeMap typeMap = mapper.createTypeMap(EmployeeDto.class, Employee .class).addMappings(mapper -> mapper.using(toUppercase).map(EmployeeDto::getName, Employe e::setFirstName));

typeMap.map(employeeDto);

* **New propertyMap**

ModelMapper mapper = new ModelMapper();

Mapper.addMapping(new PropertyMap<Employee, EmployeeDto>() {

@Override

Protected void configure() {

Map().setIncome(source.getSalary());}

**}**

return mapper.map(this.employeeRepo.findById(id), ManagerDto.class);

1. **Validation – добавя се един път в проекта**

**-**@Component  
public class ValidationUtilImpl implements ValidationUtil{  
   
 private final Validator validator;  
  
 public ValidationUtilImpl() {  
 validator = Validation.*buildDefaultValidatorFactory*().getValidator();  
 }  
  
 @Override  
 public <E> Set<ConstraintViolation<E>> violation(E entity) {  
 return validator.validate(entity);  
 }  
  
 @Override  
 public <E> boolean isValid(E entity) {  
 return validator.validate(entity).isEmpty();  
 }

**-validation annotation –** анотациите се поставят на DTO класове само на **getters**

@Pattern(regexp = “”) – дефиниране на regexp за имейла

@NotBlank – използва се за String fields, вместо NotNull

Min, Max, Size, DecimalMin, Email, message = ,

@Email(message = "Incorrect email.")  
public String getEmail() {  
 return email;  
}

* В Service се инджектва Validationutil и ModelMapper, и се валидира с метод .violation()
* Set<ConstraintViolation<UserRegisterDto>> violation = validationUtil.violation(userRegisterDto);  
  if (!violation.isEmpty()) {  
   violation  
   .stream()  
   .map(ConstraintViolation::getMessage)  
   .forEach(System.*out*::println);  
   return;  
  }
* **Unique field constraint in DTO - validation**

-проверява дали е валидно и дали не съществува такова поле в репозиторито. Ако съществува няма да го добави. Проверката ще се изпълни, когато се извика терминираща операция, например .forEach().

