1. **GSON dependency се импортира от maven**

* Анотации **@Expose** се поставят на Entity-тата, които искаме да се мапнат. (да се запише/прочете в/от класа). Поставя се САМО на полета
* Поставят се анотации за валидации в класа на DTO
* @SerializedName(value = "country") – променя името на полето, което ще чете от JSON

1. **Конфигуране на GSON**

@Bean  
public Gson gson() {  
 return new GsonBuilder()  
 .excludeFieldsWithoutExposeAnnotation()  
 .setPrettyPrinting()  
 .create();  
}

1. **Auto-Mapping multi DTO**

**-**

**При мапване с modelMapper може да се мапне към DTO, което в себе си съдръжа друг Set<OtherDto> и ако имената са еднакви, ще се напами автоматично**

1. **Четене от JSON**

* Четене на много файлове под формата на масив от DTO
* String fileContent = Files  
   .*readString*(Path.*of*(*RESOURCES\_ROOT\_PATH* + *USERS\_FILE\_NAME*));  
    
  UserSeedDto[] userSeedDto = gson.fromJson(fileContent, UserSeedDto[].class);  
    
    
  Arrays.*stream*(userSeedDto)  
   .filter(validationUtil::isValid)  
   .map(user -> modelMapper.map(user, User.class))  
   .forEach(userRepository::save);

ProductSeedDto[] productSeedDto = gson.fromJson(fileContent, ProductSeedDto[].class);  
  
Arrays.*stream*(productSeedDto)  
 .filter(validationUtil::isValid)  
 .map(product -> {  
 Product productToAdd = modelMapper.map(product, Product.class);  
 productToAdd.setSeller(userService.findRandomUser());  
 if (product.getPrice().compareTo(BigDecimal.*valueOf*(900)) > 0) {  
 productToAdd.setBuyer(userService.findRandomUser());  
 }  
  
 Set<Category> categories = categoryService.findRandomCategories();  
 productToAdd.setCategories(categories);  
  
 return productToAdd;  
 })  
 .forEach(productRepository::save);

1. **Писане в .json файл**

* С **FileWriter**
* За да запише във файла, се извикват методи .flush() и .close();

FileWriter writer = new FileWriter(path);  
  
gson.toJson(productsDto, writer);  
  
writer.flush();  
writer.close();

* С **Files.write**
* String-a се използва с Colletions.singelton(String str)
* String jsonContent = gson.toJson(categoryDto);  
    
  Files.*write*(Path.*of*(*RESOURCES\_ROOT\_PATH\_OUT* + *CATEGORIES\_BY\_PRODUCTS\_NAME*),  
   Collections.*singleton*(jsonContent));

1. **Когато се добавят обекти в <Set> се имплементира equals() и hashCode() методи.**
2. **Remove elements from Set of Objects in other Object with stream**

List<UserWithSoldProductsDto> usersDto = users.stream()  
 .map(user -> {  
 user.getSoldProducts().removeIf(p -> p.getBuyer() == null);  
 return user;  
 })  
 .map(user -> modelMapper.map(user, UserWithSoldProductsDto.class))  
 .collect(Collectors.*toList*());

1. **Създаване на DTO Обект чрез резултат от Query в Repository**

@Query("SELECT new com.example.jsonexercise.model.dto.CategoryByProductsCount(c.name, count(\*), avg(p.price), sum(p.price))" +  
 " FROM Category c JOIN c.products p GROUP BY c.id ORDER BY count(\*) DESC")  
Set<CategoryByProductsCount> findAllByOrderByProductsSize();

1. **Query, ordered by SIZE(u.soldProducts)**

@Query("SELECT u FROM User u JOIN u.soldProducts sp WHERE sp.buyer IS NOT NULL ORDER BY SIZE(u.soldProducts), u.lastName ASC")  
List<User> findAllByUsersWithAtLeastOneSoldProductOrderedByCountOfProducts();