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

* Анотации **@XmlWrapperElement, @XmlElement, @XmlAtribute** се поставят в 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
* На класа се поставят анотации:

@XmlRootElement(name = “ “)

@XmlAccessorType(XmlAccessType.FIELD)

* . В класовете винаги трябва да се дава и празен конструктор ,за да не дава грешка JAXB Exception (nor any of super class is known to this constructor)
* Могат да се добавят няколко класа, при инстанцирането на JAXBContext
* Винаги се добавя празен конструктор без аргументи

На Nest-натите класове се поставя име на RootElement, така че да съотвества на елемента от от основния клас

<dependency>  
 <groupId>javax.xml.bind</groupId>  
 <artifactId>jaxb-api</artifactId>  
 <version>2.3.1</version>  
</dependency>  
  
<dependency>  
 <groupId>com.sun.xml.bind</groupId>  
 <artifactId>jaxb-impl</artifactId>  
 <version>2.3.4</version>  
</dependency>

1. **Инициализация на JAXB context**

* masrhal

jaxbContext = JAXBContext.*newInstance*(tClass);  
Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();  
return (T)unmarshaller.unmarshal(new FileReader(filePath));

* unmarshal
* jaxbContext = JAXBContext.*newInstance*(entity.getClass());  
  Marshaller marshaller = jaxbContext.createMarshaller();  
  marshaller.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, true);  
    
  marshaller.marshal(entity, new File(filePath));

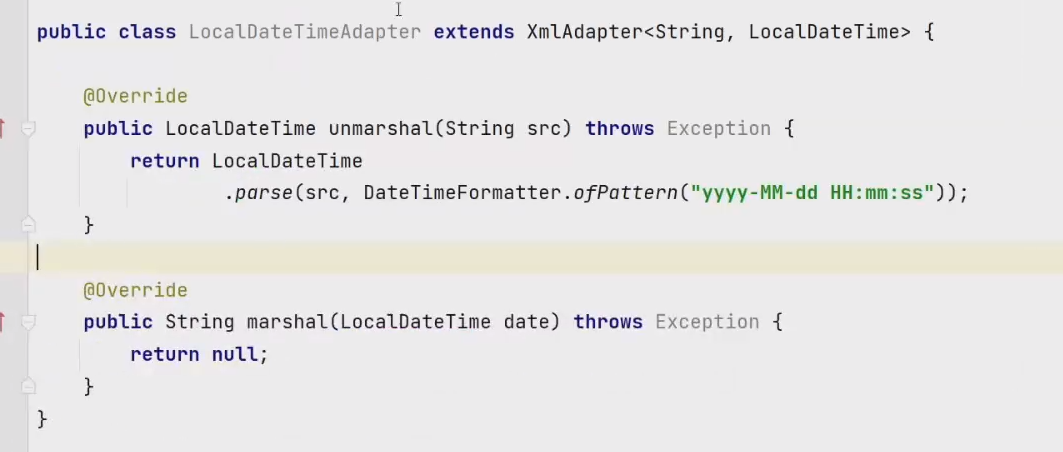
1. **Unmarashall from string (ByteArray)**

String input = “---//----“;

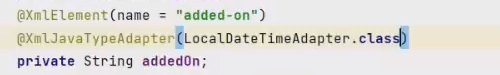
T unmarshal = (T) umarshaller.unmarshal(new ByteArrayInputStream(input.getBytes);

1. **XmlAdapter**

* Може да мапва двупосочно един тип към друг, например:
* LocalDateTime -> String и String -> LocalDateTime



* Добавя се анотация @XmlJavaTypeAdapter, който ще мапне String Към LocalDateTime



1. **Example – nested XML**

TaskSeedRootDto taskSeedRootDto = xmlParser.fromFile(*TASKS\_FILE\_PATH*, TaskSeedRootDto.class);  
  
List<TaskSeedDto> collect = taskSeedRootDto.getTasks()  
 .stream()  
 .filter(task -> {  
 boolean isValid = validationUtil.isValid(task);  
 if (!mechanicsNames.contains(task.getMechanicId().getFirstName())) {  
 isValid = false;  
 }  
 mechanicsNames.add(task.getMechanicId().getFirstName());  
  
 return isValid;  
 })  
 .map(task -> modelMapper.map(task, TaskSeedDto.class))  
 .collect(Collectors.*toList*());