**Java JAXB**

*last modified July 8, 2023*

Java JAXB tutorial shows how to use JAXB library to work with XML. The examples write Java objects into XML files and read XML data into Java objects.

**JAXB**

*Java Architecture for XML Binding (JAXB)* is a software framework that allows Java developers to map Java classes to XML representations. JAXB enables to marshal Java objects into XML and unmarshal XML back into Java objects.

In Java 9, JAXB has moved into a separate module java.xml. In Java 9 and Java 10 we need to use the --add-modules=java.xml.bind option. In Java 11, JAXB has been removed from JDK and we need to add it to the project as a separate library via Maven or Gradle.

In our examples, we use JDK 11 and Maven to create our applications.

**JAXB definitions**

*Marshalling* is the process of transforming Java objects into XML documents. *Unmarshalling* is the process of reading XML documents into Java objects. The JAXBContext class provides the client's entry point to the JAXB API. It provides API for marshalling, unmarshalling and validating.

**JAXB POM settings**

The following POM file includes the necessary JAXB JARs.

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JavaWriteXmlJaxbEx</groupId>

<artifactId>JavaWriteXmlJaxbEx</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.2.11</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-core</artifactId>

<version>2.2.11</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.2.11</version>

</dependency>

<dependency>

<groupId>javax.activation</groupId>

<artifactId>activation</artifactId>

<version>1.1.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass>com.zetcode.JavaWriteXmlJaxbEx</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

</project>

In addition to including JAXB dependencies, we use the maven-assembly-plugin to pack all the dependencies into one JAR.

**JAXB write XML example**

In the first example, we write Java objects into an XML file.

**Book.java**

package com.zetcode;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "book")

// Defining order

@XmlType(propOrder = { "author", "name", "publisher", "isbn" })

public class Book {

private String name;

private String author;

private String publisher;

private String isbn;

// Changing to title

@XmlElement(name = "title")

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public String getPublisher() {

return publisher;

}

public void setPublisher(String publisher) {

this.publisher = publisher;

}

public String getIsbn() {

return isbn;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

@Override

public String toString() {

final StringBuilder sb = new StringBuilder("Book{");

sb.append("name='").append(name).append('\'');

sb.append(", author='").append(author).append('\'');

sb.append(", publisher='").append(publisher).append('\'');

sb.append(", isbn='").append(isbn).append('\'');

sb.append('}');

return sb.toString();

}

}

This is the Book bean. This bean is going to be transformed into a specific XML tag.

@XmlRootElement(name = "book")

With the @XmlRootElement annotation, we define the XML tag name.

@XmlType(propOrder = { "author", "name", "publisher", "isbn" })

With the @XmlType's propOrder attribute we define the order of the subelements.

@XmlElement(name = "title")

public String getName() {

return name;

}

We can change the default element name to title.

**BookStore.java**

package com.zetcode;

import java.util.ArrayList;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlElementWrapper;

import javax.xml.bind.annotation.XmlRootElement;

//This statement means that class "Bookstore.java" is the root-element of our example

@XmlRootElement(namespace = "com.zetcode")

public class BookStore {

// XmLElementWrapper generates a wrapper element around XML representation

@XmlElementWrapper(name = "bookList")

// XmlElement sets the name of the entities

@XmlElement(name = "book")

private ArrayList<Book> bookList;

private String name;

private String location;

public void setBookList(ArrayList<Book> bookList) {

this.bookList = bookList;

}

public ArrayList<Book> getBooksList() {

return bookList;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getLocation() {

return location;

}

public void setLocation(String location) {

this.location = location;

}

}

The BookStore is a class that contains a list where we place our book objects.

@XmlRootElement(namespace = "com.zetcode")

public class BookStore {

We define the root element with the @XmlRootElement annotation.

// XmLElementWrapper generates a wrapper element around XML representation

@XmlElementWrapper(name = "bookList")

// XmlElement sets the name of the entities

@XmlElement(name = "book")

private ArrayList<Book> bookList;

The @XmlElementWrapper annotation defines a wrapper element around the book elements. The @XmlElement annotation defines the name of the XML element that goes inside the wrapper.

**JavaWriteXmlJaxbEx.java**

package com.zetcode;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Marshaller;

import java.io.File;

import java.util.ArrayList;

public class JavaWriteXmlJaxbEx {

private static final String BOOKSTORE\_XML = "src/main/resources/bookstore.xml";

public static void main(String[] args) throws JAXBException {

var bookList = new ArrayList<Book>();

// create books

var book1 = new Book();

book1.setIsbn("978-0060554736");

book1.setName("The Game");

book1.setAuthor("Neil Strauss");

book1.setPublisher("Harpercollins");

bookList.add(book1);

var book2 = new Book();

book2.setIsbn("978-3832180577");

book2.setName("Feuchtgebiete");

book2.setAuthor("Charlotte Roche");

book2.setPublisher("Dumont Buchverlag");

bookList.add(book2);

// create bookstore, assign books

var bookstore = new BookStore();

bookstore.setName("Fraport Bookstore");

bookstore.setLocation("Livres belles");

bookstore.setBookList(bookList);

// create JAXB context and instantiate marshaller

var context = JAXBContext.newInstance(BookStore.class);

var m = context.createMarshaller();

m.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);

// Write to System.out

m.marshal(bookstore, System.out);

// Write to File

m.marshal(bookstore, new File(BOOKSTORE\_XML));

}

}

In the example, we create book objects, add them to the bookstore and transform the bookstore into an XML file.

// create books

var book1 = new Book();

book1.setIsbn("978-0060554736");

book1.setName("The Game");

book1.setAuthor("Neil Strauss");

book1.setPublisher("Harpercollins");

bookList.add(book1);

var book2 = new Book();

book2.setIsbn("978-3832180577");

book2.setName("Feuchtgebiete");

book2.setAuthor("Charlotte Roche");

book2.setPublisher("Dumont Buchverlag");

bookList.add(book2);

We create two book objects.

// create bookstore, assign books

var bookstore = new BookStore();

bookstore.setName("Fraport Bookstore");

bookstore.setLocation("Livres belles");

bookstore.setBookList(bookList);

A bookstore is created and books are placed into it.

// create JAXB context and instantiate marshaller

var context = JAXBContext.newInstance(BookStore.class);

We create a new JAXBContext. We pass a list of classes that the new context object has to recognize. (In our case it is one class.)

var m = context.createMarshaller();

m.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);

From the context we get a marshaller with createMarshaller. We set a property to get formatted output.

// Write to System.out

m.marshal(bookstore, System.out);

// Write to File

m.marshal(bookstore, new File(BOOKSTORE\_XML));

We write the data into the system output and a file.

Referencias extras  
<https://www.arquitecturajava.com/introduccion-java-jaxb/>

<https://www.baeldung.com/jaxb>