 Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

**ОТЧЕТ**

по лабораторной работе №4

по дисциплине «СМиСПИС»

**«Технология XML связывания данных с помощью JAXB»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5371 |  | Мартынов М. |
| Студентка гр. 5371 |  | Козлова С. |
| Студент гр. 5371 |  | Аверкиев В. |
| Преподаватель |  | Кораблев Ю.А. |

Санкт-Петербург

2020

1. **Цель работы**

Изучить технологию XML связывания данных с помощью API связывания – JAXB.

1. **Задание на лабораторную работу №3. Вариант №3.**

Создать приложение, которое читает данные телефонного справочника из XML, а также записывает данные обратно в XML.

1. **Выполнение лабораторной работы**

**Настройка сборщика Maven**

Так как на нашем компьютере установлена Open JDK 11, а библиотека JAXB была исключена из JDK дистрибутивов начиная с JDK версии 6 и мы используем сборщик Maven, то для начала необходимо добавить Maven () зависимости на эту библиотеку.

Пропишем в pom.xml – конфигурационный файл сборщика Maven зависимости на JAXB:

<dependency>

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

<artifactId>jaxb-api</artifactId>

<version2.3.1</version>

</dependency>

<dependency>

<groupId>org.glassfish.jaxb</groupId>

<artifactId>jaxb-runtime</artifactId>

<version>2.3.1</version>

</dependency>

Так же добавим зависимости на библиотеку для Unit тестирования приложений:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<scope>test</scope>

<version>${junit.jupiter.version}</version>

</dependency>

Добавим плагины к сборщику Maven, которые будут автоматически запускать наши тестовые методы, помеченные аннотацией @Test при сборке приложения:

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

</plugin>

<plugin>

<artifactId>maven-failsafe-plugin</artifactId>

<version>2.22.2</version>

</plugin>

**Написание DTD документа валидации XML**

Был написан DTD документ telephone\_book.dtd валидации нашего XML файла:

<!DOCTYPE records [

<!ELEMENT name EMPTY>

<!ELEMENT address EMPTY>

<!ELEMENT number EMPTY>

<!ELEMENT contact (name, address, number)>

<!ELEMENT records (contact\*)>

<!ATTLIST name

first CDATA #REQUIRED

last CDATA #REQUIRED

>

<!ATTLIST address

home CDATA #REQUIRED

work CDATA #REQUIRED

>

<!ATTLIST number

home CDATA #REQUIRED

work CDATA #REQUIRED

>

]>

**Написание XSD схемы валидации XML**

Был написана схема telephone\_book\_scheme.xsd валидации нашего XML файла:

<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="https://github.com/Sindicat/krlabs/tree/master/lab4">

<element name="records">

<complexType>

<sequence>

<element name="contact" maxOccurs="unbounded" minOccurs="0">

<complexType>

<sequence>

<element name="name">

<complexType>

<attribute name="first" type="string" use="required"/>

<attribute name="last" type="string" use="required"/>

</complexType>

</element>

<element name="address">

<complexType>

<attribute name="home" type="string" use="required"/>

<attribute name="work" type="string" use="required"/>

</complexType>

</element>

<element name="number">

<complexType>

<attribute name="home" type="string" use="required"/>

<attribute name="work" type="string" use="required"/>

</complexType>

</element>

</sequence>

</complexType>

</element>

</sequence>

</complexType>

</element>

</schema>

**Вид валидного XML документа**

Ниже представлен пример валидного XML документа, содержащего данные телефонного справочника:

*<?***xml version="1.0" encoding="UTF-8" standalone="yes"***?>*<**records**>  
 <**contact**>  
 <**address home="London" work="New York"**/>  
 <**name first="Rowan" last="Gill"**/>  
 <**number home="+1-532-521-23" work="+1-533-654-12"**/>  
 </**contact**>  
 <**contact**>  
 <**address home="Denver" work="Denver"**/>  
 <**name first="Harley" last="Gibson"**/>  
 <**number home="+1-331-521-77" work="+1-939-654-32"**/>  
 </**contact**>  
 <**contact**>  
 <**address home="Waterville" work="Bedford"**/>  
 <**name first="Ashley" last="Evans"**/>  
 <**number home="+1-831-521-10" work="+1-144-654-65"**/>  
 </**contact**>  
</**records**>

**Разработка классов для отображения XML документа**

UML диаграмма разработанных классов изображена на рис.1

A screenshot of a cell phone

Description automatically generated

Рисунок 1 – UML диаграмма классов для отображения XML документа

**Тестирование маршаллизации и демаршаллизации**

Для тестирования маршаллизации и демаршаллизации был написан класс MarshallingDemarshallingTest, который содержит два метода, проаннотированные аннотацией @Test:

marshallingTest() - для тестирования маршаллизации. Метод создает предзаполненный данными класс TelephoneDirectory, затем маршаллизует его и сравнивает полученную строку в формета XML с эталонным значением из прочитанного файла telephone\_book\_test.xml.

demarshallingTest – для тестирования демаршаллизации. Метод читает XML документ telephone\_book\_test.xml, демаршаллизует его в объект TelephoneDirectory и сравнивает с эталонным объектом TelephoneDirectory, который создается программно в тестовом методе.

1. **Зачетное задание**

*Формулировка задания:* добавить в ХМL-файл тег для указания контактов, связанных с ним. Тег должен содержать ссылку на следующий контакт из списка. Если в этом поле ссылки нет, это означает конец списка. Показать возможность извлечения всего списка контактов для выбранного контакта.

**Добавление новых атрибутов в XML документ**

Были добавлены два атрибута в XML атрибут <contact>:

* *<id>* - для однозначной идентификации текущего контакта;
* *<nextContactId>* - указывается id следующего контакта.

Таким образом валидный XML документ выглядит следующим образом:

<**records**>  
 <**contact**>  
 <**address home="London" work="New York"**/>  
 <**name first="Rowan" last="Gill"**/>  
 <**number home="+1-532-521-23" work="+1-533-654-12"**/>  
 <**id**>1</**id**>  
 <**nextContactId**>3</**nextContactId**>  
 </**contact**>

<**records**>

**Добавление новых полей в класс Contact**

Были добавлены два поля:

**private** Long **id**;  
  
**private** Long **nextContactId**;

В текущие поля будут заполнять данными из добавленных атрибутов при маршаллизация XML документа.

**Добавление логики по поиску подконтактов выбранного контакта**

Добавлен новый класс TelephoneDirectoryService, который в конструкторе принимает объект телефонного словаря с контактами – TelephoneDirectory). В данном сервисе есть public метод findAllSubContacts(final Long contactId). Данная функция с помощью рекурсивного поиска находит все под контакты заданного контакта.

Ниже приведен код сервиса TelephoneDirectoryService:

**package** com.github.sindicat.lab4.dto;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.function.Function;  
**import** java.util.stream.Collectors;  
  
**public class** TelephoneDirectoryService {  
  
 **private final** Map<Long, Contact> **contactById**;  
  
 **public** TelephoneDirectoryService(**final** TelephoneDirectory telephoneDirectory) {  
 **contactById** = telephoneDirectory.getContact().stream()  
 .collect(Collectors.*toMap*(Contact::getId, Function.*identity*()));  
 }  
  
 **public** List<Contact> findAllSubContacts(**final** Long contactId) {  
 **final** Contact contact = **contactById**.get(contactId);  
 **if** (contact.getNextContactId() == **null**) {  
 **return** List.*of*();  
 } **else** {  
 **return** findSubContacts(**contactById**.get(contact.getNextContactId()));  
 }  
 }  
  
 **private** List<Contact> findSubContacts(**final** Contact contact) {  
 **final var** nextContactId = contact.getNextContactId();  
 **if** (nextContactId == **null**) {  
 **return** List.*of*(contact);  
 }  
 **final var** subContacts = **new** ArrayList<Contact>();  
 subContacts.add(contact);  
 subContacts.addAll(findSubContacts(**contactById**.get(nextContactId)));  
 **return** subContacts;  
 }  
  
}

**Тестирование добавленной функциональности**

Для тестирования новой функциональности был добавлен новый XML файл contact\_links\_test.xml. Содержимое данного файла:

*<?***xml version="1.0" encoding="UTF-8" standalone="yes"***?>*<**records**>  
 <**contact**>  
 <**address home="London" work="New York"**/>  
 <**name first="Rowan" last="Gill"**/>  
 <**number home="+1-532-521-23" work="+1-533-654-12"**/>  
 <**id**>1</**id**>  
 <**nextContactId**>3</**nextContactId**>  
 </**contact**>  
 <**contact**>  
 <**address home="Denver" work="Denver"**/>  
 <**name first="Harley" last="Gibson"**/>  
 <**number home="+1-331-521-77" work="+1-939-654-32"**/>  
 <**id**>2</**id**>  
 <**nextContactId**>1</**nextContactId**>  
 </**contact**>  
 <**contact**>  
 <**address home="Waterville" work="Bedford"**/>  
 <**name first="Ashley" last="Evans"**/>  
 <**number home="+1-831-521-10" work="+1-144-654-65"**/>  
 <**id**>3</**id**>  
 <**nextContactId**/>  
 </**contact**>  
 <**contact**>  
 <**address home="Paris" work="Paris"**/>  
 <**name first="Michel" last="Sardou"**/>  
 <**number home="+5-131-121-90" work="+5-247-034-99"**/>  
 <**id**>4</**id**>  
 <**nextContactId**>3</**nextContactId**>  
 </**contact**>  
</**records**>

И написан класс с Unit тестом новой функциональности – TelephoneDirectoryServiceTest.

Класс содержит один тестовый метод findAllSubContactsTest(), который читает файл contact\_links\_test.xml, маршаллизует содержимое файла в класс TelephoneDirectory и с помощью сервиса TelephoneDirectoryService ищет все подконтакты контакта с id = 2.

Ожидаемые значения при таком критерии поиска это подконтакты с id = 1 и id = 3.

Исходный код метода findAllSubContactsTest():

@Test  
**public void** findAllSubContactsTest() **throws** JAXBException {  
 File file = **new** File(**"src/test/resources/data/contact\_links\_test.xml"**);  
 JAXBContext jaxbContext = JAXBContext.*newInstance*(TelephoneDirectory.**class**);  
  
 **final var** jaxbUnmarshaller = jaxbContext.createUnmarshaller();  
 **final var** telephoneDirectory = (TelephoneDirectory) jaxbUnmarshaller.unmarshal(file);  
 **final var** telephoneDirectoryService = **new** TelephoneDirectoryService(telephoneDirectory);  
 **final** List<Contact> subContacts = telephoneDirectoryService.findAllSubContacts(2L);  
  
 Assert.*assertEquals*(2, subContacts.size());  
 Assert.*assertEquals*(Long.*valueOf*(1L), subContacts.get(0).getId());  
 Assert.*assertEquals*(Long.*valueOf*(3L), subContacts.get(1).getId());  
}

1. **Исходный код**

**Класс Address**

**package** com.github.sindicat.lab4.dto.contact;  
  
**import** javax.xml.bind.annotation.XmlAttribute;  
**import** javax.xml.bind.annotation.XmlType;  
**import** java.util.Objects;  
  
@XmlType(name = **"address"**)  
**public class** Address {  
  
 **private** String **home**;  
  
 **private** String **work**;  
  
 **public** Address() {  
 }  
  
 **public** Address(**final** String home, **final** String work) {  
 **this**.**home** = home;  
 **this**.**work** = work;  
 }  
  
 @XmlAttribute(name = **"home"**)  
 **public** String getHome() {  
 **return home**;  
 }  
  
 **public void** setHome(**final** String home) {  
 **this**.**home** = home;  
 }  
  
 @XmlAttribute(name = **"work"**)  
 **public** String getWork() {  
 **return work**;  
 }  
  
 **public void** setWork(**final** String work) {  
 **this**.**work** = work;  
 }  
  
 @Override  
 **public boolean** equals(**final** Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 **final** Address address = (Address) o;  
 **return** Objects.*equals*(**home**, address.**home**) &&  
 Objects.*equals*(**work**, address.**work**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**home**, **work**);  
 }  
}

**Класс Name**

**package** com.github.sindicat.lab4.dto.contact;  
  
**import** javax.xml.bind.annotation.XmlAttribute;  
**import** javax.xml.bind.annotation.XmlType;  
**import** java.util.Objects;  
  
@XmlType(name = **"name"**)  
**public class** Name {  
  
 **private** String **first**;  
  
 **private** String **last**;  
  
 **public** Name() {  
 }  
  
 **public** Name(**final** String first, **final** String last) {  
 **this**.**first** = first;  
 **this**.**last** = last;  
 }  
  
 @XmlAttribute(name = **"first"**)  
 **public** String getFirst() {  
 **return first**;  
 }  
  
 **public void** setFirst(**final** String first) {  
 **this**.**first** = first;  
 }  
  
 @XmlAttribute(name = **"last"**)  
 **public** String getLast() {  
 **return last**;  
 }  
  
 **public void** setLast(**final** String last) {  
 **this**.**last** = last;  
 }  
  
 @Override  
 **public boolean** equals(**final** Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 **final** Name name = (Name) o;  
 **return** Objects.*equals*(**first**, name.**first**) &&  
 Objects.*equals*(**last**, name.**last**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**first**, **last**);  
 }  
}

**Класс Number**

**package** com.github.sindicat.lab4.dto.contact;  
  
  
**import** javax.xml.bind.annotation.XmlAttribute;  
**import** javax.xml.bind.annotation.XmlType;  
**import** java.util.Objects;  
  
@XmlType(name = **"number"**)  
**public class** Number {  
  
 **private** String **home**;  
  
 **private** String **work**;  
  
 **public** Number() {  
 }  
  
 **public** Number(**final** String home, **final** String work) {  
 **this**.**home** = home;  
 **this**.**work** = work;  
 }  
  
 @XmlAttribute(name = **"home"**)  
 **public** String getHome() {  
 **return home**;  
 }  
  
 **public void** setHome(**final** String home) {  
 **this**.**home** = home;  
 }  
  
 @XmlAttribute(name = **"work"**)  
 **public** String getWork() {  
 **return work**;  
 }  
  
 **public void** setWork(**final** String work) {  
 **this**.**work** = work;  
 }  
  
 @Override  
 **public boolean** equals(**final** Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 **final** Number number = (Number) o;  
 **return** Objects.*equals*(**home**, number.**home**) &&  
 Objects.*equals*(**work**, number.**work**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**home**, **work**);  
 }  
}

**Класс Contact**

**package** com.github.sindicat.lab4.dto;  
**import** com.github.sindicat.lab4.dto.contact.Address;  
**import** com.github.sindicat.lab4.dto.contact.Name;  
**import** com.github.sindicat.lab4.dto.contact.Number;  
**import** javax.xml.bind.annotation.XmlType;  
**import** java.util.Objects;  
  
@XmlType(name = **"contact"**)  
**public class** Contact {  
  
 **private** Name **name**;  
 **private** Number **number**;  
 **private** Address **address**;  
  
 **public** Contact() {  
 }  
 **public** Contact(**final** Name name, **final** Number number, **final** Address address) {  
 **this**.**name** = name;  
 **this**.**number** = number;  
 **this**.**address** = address;  
 }  
  
 **public** Name getName() {  
 **return name**;  
 }  
  
 **public void** setName(**final** Name name) {  
 **this**.**name** = name;  
 }  
  
 **public** Number getNumber() {  
 **return number**;  
 }  
  
 **public void** setNumber(**final** Number number) {  
 **this**.**number** = number;  
 }  
  
 **public** Address getAddress() {  
 **return address**;  
 }  
  
 **public void** setAddress(**final** Address address) {  
 **this**.**address** = address;  
 }  
 @Override  
 **public boolean** equals(**final** Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 **final** Contact contact = (Contact) o;  
 **return** Objects.*equals*(**name**, contact.**name**) &&  
 Objects.*equals*(**number**, contact.**number**) &&  
 Objects.*equals*(**address**, contact.**address**);  
 }  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**name**, **number**, **address**);  
 }  
}

**Класс TelephoneDirectory**

**package** com.github.sindicat.lab4.dto;  
  
**import** javax.xml.bind.annotation.XmlRootElement;  
**import** java.util.List;  
**import** java.util.Objects;  
  
@XmlRootElement(name = **"records"**)  
**public class** TelephoneDirectory {  
  
 **private** List<Contact> **contact**;  
  
 **public** TelephoneDirectory() {  
 }  
  
 **public** TelephoneDirectory(**final** List<Contact> contact) {  
 **this**.**contact** = contact;  
 }  
  
 **public** List<Contact> getContact() {  
 **return contact**;  
 }  
  
 **public void** setContact(**final** List<Contact> contact) {  
 **this**.**contact** = contact;  
 }  
  
 @Override  
 **public boolean** equals(**final** Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 **final** TelephoneDirectory that = (TelephoneDirectory) o;  
 **return** Objects.*equals*(**contact**, that.**contact**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**contact**);  
 }  
}

**Класс MarshallingDemarshallingTest**

**package** com.github.sindicat.lab4;  
  
**import** com.github.sindicat.lab4.dto.Contact;  
**import** com.github.sindicat.lab4.dto.TelephoneDirectory;  
**import** com.github.sindicat.lab4.dto.contact.Address;  
**import** com.github.sindicat.lab4.dto.contact.Name;  
**import** com.github.sindicat.lab4.dto.contact.Number;  
**import** org.junit.Assert;  
**import** org.junit.jupiter.api.Test;  
  
**import** javax.xml.bind.JAXBContext;  
**import** javax.xml.bind.JAXBException;  
**import** javax.xml.bind.Marshaller;  
**import** javax.xml.bind.Unmarshaller;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.io.StringWriter;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.Arrays;  
**import** java.util.List;  
**import** java.util.stream.Collectors;  
  
**public class** MarshallingDemarshallingTest {  
  
 @Test  
 **public void** marshallingTest() **throws** JAXBException, IOException {  
 **final** TelephoneDirectory telephoneDirectory = createTestTelephoneDirectory();  
  
 JAXBContext context = JAXBContext.*newInstance*(TelephoneDirectory.**class**);  
 Marshaller marshaller = context.createMarshaller();  
 marshaller.setProperty(Marshaller.***JAXB\_FORMATTED\_OUTPUT***, Boolean.***TRUE***);  
  
 StringWriter writer = **new** StringWriter();  
 marshaller.marshal(telephoneDirectory, writer);  
  
 List<String> demarshallingResultByLine = Arrays.*stream*(writer.toString().split(**"\n"**))  
 .map(line -> line.trim())  
 .collect(Collectors.*toList*());  
  
 Path pathToTestFile = Paths.*get*(**"src/test/resources/data/telephone\_book\_test.xml"**);  
 List<String> expectedValue = Files.*readAllLines*(pathToTestFile)  
 .stream()  
 .map(line -> line.trim())  
 .collect(Collectors.*toList*());  
  
 Assert.*assertEquals*(expectedValue, demarshallingResultByLine);  
 }  
  
 @Test  
 **public void** demarshallingTest() **throws** JAXBException, IOException {  
 File file = **new** File(**"src/test/resources/data/telephone\_book\_test.xml"**);  
 JAXBContext jaxbContext = JAXBContext.*newInstance*(TelephoneDirectory.**class**);  
  
 Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();  
 TelephoneDirectory actualTelephoneDirectory = (TelephoneDirectory) jaxbUnmarshaller.unmarshal(file);  
 TelephoneDirectory expectedTelephoneDirectory = createTestTelephoneDirectory();  
  
 Assert.*assertEquals*(expectedTelephoneDirectory, actualTelephoneDirectory);  
 }  
  
 **private** TelephoneDirectory createTestTelephoneDirectory() {  
 **final var** name1 = **new** Name(**"Rowan"**, **"Gill"**);  
 **final var** address1 = **new** Address(**"London"**, **"New York"**);  
 **final var** number1 = **new** Number(**"+1-532-521-23"**, **"+1-533-654-12"**);  
 **final var** contact1 = **new** Contact(name1, number1, address1);  
  
 **final var** name2 = **new** Name(**"Harley"**, **"Gibson"**);  
 **final var** address2 = **new** Address(**"Denver"**, **"Denver"**);  
 **final var** number2 = **new** Number(**"+1-331-521-77"**, **"+1-939-654-32"**);  
 **final var** contact2 = **new** Contact(name2, number2, address2);  
  
 **final var** name3 = **new** Name(**"Ashley"**, **"Evans"**);  
 **final var** address3 = **new** Address(**"Waterville"**, **"Bedford"**);  
 **final var** number3 = **new** Number(**"+1-831-521-10"**, **"+1-144-654-65"**);  
 **final var** contact3 = **new** Contact(name3, number3, address3);  
  
 **return new** TelephoneDirectory(  
 List.*of*(  
 contact1,  
 contact2,  
 contact3  
 )  
 );  
 }  
}

1. **Вывод**

В процессе выполнения лабораторной работыбыл изучен формат данных XML, а так же способы его валидации. Изучены маршлаллизация и демаршаллизация Java объектов с помощью библиотеки JAXB.