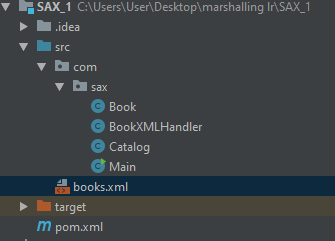
**Задание №3 Marshalling**

1. **SAX** парсер.

Создадим структуру проекта:



Нам потребуется класс Book, который описывает на объект маршализации.

BookXMLHandler, необходимый для бизнес логики и методов чтения их файла.

Catalog, для описания модели содержания всех книг.

Main, для создания парсера данных.

Добавим данные в xml файл:

<?xml version="1.0"?>  
<catalog>  
 <book id="bk101">  
 <author>Matthew</author>  
 <title>XML Developer's Guide</title>  
 <genre>Computer</genre>  
 <price>44.95</price>  
 <publish\_date>2000-10-01</publish\_date>  
 <description>An in-depth look at creating applications  
 with XML.</description>  
 </book>  
 <book id="bk102">  
 <author>Ralls, Kim</author>  
 <title>Midnight Rain</title>  
 <genre>Fantasy</genre>  
 <price>5.95</price>  
 <publish\_date>2000-12-16</publish\_date>  
 <description>A former architect battles corporate zombies,  
 an evil sorceress, and her own childhood to become queen  
 of the world.</description>  
 </book>  
  
</catalog>

Класс Main:

package com.sax;  
  
import org.xml.sax.SAXException;  
  
import javax.xml.parsers.ParserConfigurationException;  
import javax.xml.parsers.SAXParser;  
import javax.xml.parsers.SAXParserFactory;  
import java.io.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
 String path = new File("C:/Users/User/Desktop/SAX\_1/src/books.xml").getAbsolutePath();  
 try {  
 SAXParserFactory factory = SAXParserFactory.*newInstance*();  
 SAXParser parser = factory.newSAXParser();  
 BookXMLHandler handler = new BookXMLHandler();  
 parser.parse(new File(path), handler);  
 } catch (IOException x) {  
 System.*err*.println(x);  
 } catch (ParserConfigurationException e) {  
 e.printStackTrace();  
 } catch (SAXException e) {  
 e.printStackTrace();  
 }  
 }  
}

Данный класс содержит основу для создания парсера и перенаправление на обработку и чтения данных из XML.

Класс Catalog:

package com.sax;  
  
import java.util.ArrayList;  
  
  
public class Catalog {  
 private ArrayList<Book> books;  
  
 public Catalog() {  
 books = new ArrayList<>();  
 }  
  
 public ArrayList<Book> getBooks() {  
 return books;  
 }  
  
 public void setBooks(ArrayList<Book> books) {  
 this.books = books;  
 }  
  
 public void push(Book b) {  
 books.add(b);  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder sb = new StringBuilder();  
 for (Book b : books) {  
 sb.append(b.toString());  
 }  
 return sb.toString();  
 }  
}

Данный класс содержит коллекцию для хранения данных полученных их XML, getters\setters. Так же метод push() для добавления данных из другого класса. Метод toString для вывода содержимого коллекции в строковом виде.

Класс Book:

package com.sax;  
  
public class Book {  
 private String id;  
 private String author;  
 private String title;  
 private String genre;  
 private Double price;  
 private String publishDate;  
 private String description;  
  
 public String getAuthor() {  
 return author;  
 }  
  
 public void setAuthor(String author) {  
 this.author = author;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getGenre() {  
 return genre;  
 }  
  
 public void setGenre(String genre) {  
 this.genre = genre;  
 }  
  
 public Double getPrice() {  
 return price;  
 }  
  
 public void setPrice(Double price) {  
 this.price = price;  
 }  
  
 public String getPublishDate() {  
 return publishDate;  
 }  
  
 public void setPublishDate(String publishDate) {  
 this.publishDate = publishDate;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString() {  
 return getTitle() + " " + getAuthor() + " " + getDescription();  
 }  
}

Класс POJO для представления объекта модели.

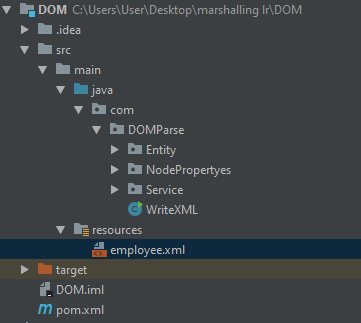
Класс BookXMLHandler:

package com.sax;  
  
import org.xml.sax.Attributes;  
import org.xml.sax.helpers.DefaultHandler;  
  
  
public class BookXMLHandler extends DefaultHandler {  
 Catalog myCatalog = null;  
 Book book = null;  
  
 boolean author = false;  
 boolean title = false;  
 boolean description = false;  
 String currentCharacters;  
  
 @Override  
 public void startElement(String uri, String localName, String qName, Attributes attributes) {  
 if (qName.equalsIgnoreCase("catalog")) {  
 myCatalog = new Catalog();  
 } else if (qName.equalsIgnoreCase("book")) {  
 book = new Book();  
 } else if (qName.equalsIgnoreCase("author")) {  
 author = true;  
 } else if (qName.equalsIgnoreCase("title")) {  
 title = true;  
 } else if (qName.equalsIgnoreCase("description")) {  
 description = true;  
 }  
 }  
  
 @Override  
 public void endElement(String uri, String localName, String qName) {  
 if (qName.equalsIgnoreCase("catalog")) {  
 System.*out*.println("Done with catalog");  
 System.*out*.println(myCatalog.toString());  
 } else if (qName.equalsIgnoreCase("book")) {  
 myCatalog.push(book);  
 System.*out*.println(book);  
 } else if (qName.equalsIgnoreCase("author")) {  
 book.setAuthor(currentCharacters);  
 author = false;  
 } else if (qName.equalsIgnoreCase("title")) {  
 book.setTitle(currentCharacters);  
 title = false;  
 } else if (qName.equalsIgnoreCase("description")) {  
 book.setDescription(currentCharacters);  
 description = false;  
 }  
 }  
  
 @Override  
 public void characters(char ch[], int start, int length ) {  
 currentCharacters = new String(ch, start, length);  
 }  
}

Данный класс необходим для чтения данных из xml. Может содержать ряд методов.

1. **DOM** парсер.

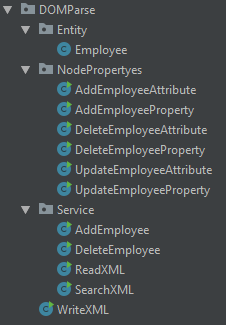
Создадим структуру проекта:



Добавим данные в xml:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<Employees>  
 <Employee id="1" qwe="22" yes="n">  
 <name>Rohit</name>  
 <gender>male</gender>  
 <age>24</age>  
 <role>software</role>  
 </Employee>  
 <Employee id="2">  
 <name>shivani</name>  
 <gender>male</gender>  
 <age>19</age>  
 <role>engineer</role>  
 </Employee>  
 <Employee id="3">  
 <name>dima</name>  
 <gender>male</gender>  
 <age>23</age>  
 <role>admin</role>  
 </Employee>  
</Employees>

Структура классов, будет следующая:



В первую очередь нам понадобится POJO класс для описания объекта читаемого из xml.

Класс Employee:

package com.DOMParse.Entity;  
  
public class Employee {  
  
 private int id;  
 private String name;  
 private int age;  
 private String gender;  
 private String role;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public String getGender() {  
 return gender;  
 }  
  
 public void setGender(String gender) {  
 this.gender = gender;  
 }  
  
 public String getRole() {  
 return role;  
 }  
  
 public void setRole(String role) {  
 this.role = role;  
 }  
  
 @Override  
 public String toString() {  
 return "Employee[id=" + getId() + ", name=" + getName() + ", gender=" + getGender() + ", age=" + getAge() + ", role=" + getRole() + "]";  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
}

Опять же обычный класс для представления объектов.

Рассмотрим слой Service. Все классы будут содержать общую конструкцию создания парсера.

Класс AddEmployee

package com.DOMParse.Service;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
import javax.xml.transform.OutputKeys;  
import javax.xml.transform.Transformer;  
import javax.xml.transform.TransformerException;  
import javax.xml.transform.TransformerFactory;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
  
import com.DOMParse.Entity.Employee;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.NodeList;  
import org.xml.sax.SAXException;  
  
public class AddEmployee {  
  
 public static void main(String[] args) throws IOException, ParserConfigurationException, SAXException, TransformerException {  
  
 File file = new File("src/main/resources/employee.xml");  
 BufferedReader input = null;  
 try {  
 input = new BufferedReader(new InputStreamReader(System.*in*));  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(file);  
 System.*out*.println("Enter name : ");  
 String name = input.readLine();  
 System.*out*.println("Enter gender : (male/female)");  
 String gender = input.readLine();  
 System.*out*.println("Enter age : ");  
 int age = Integer.*parseInt*(input.readLine());  
 System.*out*.println("Enter role : ");  
 String role = input.readLine();  
 Employee employee = new Employee();  
 employee.setName(name);  
 employee.setGender(gender);  
 employee.setAge(age);  
 employee.setRole(role);  
 Element element = *getEmployeeNode*(employee, document);  
 document.getDocumentElement().appendChild(element);  
 TransformerFactory tFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = tFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = new DOMSource(document);  
 StreamResult sResult = new StreamResult(file);  
 transformer.transform(source, sResult);  
 System.*out*.println("Employee has been added successfully.");  
 } finally {  
 if (input != null) {  
 input.close();  
 }  
 }  
  
 }  
   
 private static Element getEmployeeNode(Employee employee, Document document) {  
  
 Element element = document.createElement("Employee");  
 NodeList list=document.getElementsByTagName("Employee");  
 int count=list.getLength();  
 element.setAttribute("id", String.*valueOf*(++count));  
  
 Element name = *getPropertyNode*("name", document, employee.getName());  
 element.appendChild(name);  
 Element gender = *getPropertyNode*("gender", document, employee.getGender());  
 element.appendChild(gender);  
 Element age = *getPropertyNode*("age", document, String.*valueOf*(employee.getAge()));  
 element.appendChild(age);  
 Element role = *getPropertyNode*("role", document, employee.getRole());  
 element.appendChild(role);  
 return element;  
 }  
  
 private static Element getPropertyNode(String property, Document document, String value) {  
  
 Element element = document.createElement(property);  
 element.setTextContent(value);  
 return element;  
 }  
  
}

Во-первых передается путь к нашему файлу. Далее создается documentBuilder для работы с xml.

Класс DeleteEmployee:

package com.DOMParse.Service;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
import javax.xml.transform.OutputKeys;  
import javax.xml.transform.Transformer;  
import javax.xml.transform.TransformerException;  
import javax.xml.transform.TransformerFactory;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
import org.xml.sax.SAXException;  
  
public class DeleteEmployee {  
  
 public static void main(String[] args) throws IOException,ParserConfigurationException, SAXException, TransformerException {  
  
 File file = new File("src/main/resources/employee.xml");  
 BufferedReader input = null;  
 try {  
 input = new BufferedReader(new InputStreamReader(System.*in*));  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(file);  
 System.*out*.println("Enter employee id : ");  
 int id = Integer.*parseInt*(input.readLine());  
 boolean result = *deleteEmployeeFromXml*(document, id);  
 if (result) {  
 TransformerFactory tFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = tFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = new DOMSource(document);  
 StreamResult sResult = new StreamResult(file);  
 transformer.transform(source, sResult);  
 System.*out*.println("Employee has been deleted successfully.");  
 } else {  
 System.*out*.println("Employee not exist.");  
 }  
 } finally {  
 if (input != null) {  
 input.close();  
 }  
 }  
  
 }  
  
  
 private static boolean deleteEmployeeFromXml(Document document, int id){  
  
 NodeList list = document.getElementsByTagName("Employee");  
 boolean result = false;  
 int length = list.getLength();  
 for (int i = 0; i < length; i++) {  
 Node node = list.item(i);  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 if (element.getAttribute("id").equals(String.*valueOf*(id))) {  
 Node prev = node.getPreviousSibling();  
 if (prev != null && prev.getNodeType() == Node.*TEXT\_NODE* && prev.getNodeValue().trim().length() == 0) {  
 document.getDocumentElement().removeChild(prev);  
 }  
 document.getDocumentElement().removeChild(element);  
 result=true;  
 break;  
 }  
  
 }  
 }  
  
 return result;  
  
 }  
  
}

Класс ReadXML:

package com.DOMParse.Service;  
  
import java.io.File;  
import java.util.ArrayList;  
import java.util.List;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
  
import com.DOMParse.Entity.Employee;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
  
public class ReadXML {  
  
 public static void main(String[] args) {  
  
 File file = new File("src/main/resources/employee.xml");  
 try {  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(file);  
 document.getDocumentElement().normalize();  
 String rootNode = document.getDocumentElement().getNodeName();  
 System.*out*.println("Root Element : " + rootNode);  
 List<Employee> employees = *getEmployeesData*(document);  
 for (Employee emp : employees) {  
 System.*out*.println(emp);  
 }  
  
 } catch (Exception exception) {  
 exception.printStackTrace();  
 }  
  
 }  
  
 private static List<Employee> getEmployeesData(Document document) {  
  
 NodeList list = document.getElementsByTagName("Employee");  
 int length = list.getLength();  
 List<Employee> employees = new ArrayList<>();  
 for (int i = 0; i < length; i++) {  
 Node node = list.item(i);  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 Employee emp = *getEmployee*(element);  
 employees.add(emp);  
 }  
 }  
  
 return employees;  
 }  
  
 private static Employee getEmployee(Element element) {  
  
 String id=element.getAttribute("id");  
 String name = *getEmployeeDetails*(element, "name");  
 String gender = *getEmployeeDetails*(element, "gender");  
 int age = Integer.*parseInt*(*getEmployeeDetails*(element, "age"));  
 String role = *getEmployeeDetails*(element, "role");  
 Employee emp = new Employee();  
 emp.setId(Integer.*parseInt*(id));  
 emp.setName(name);  
 emp.setAge(age);  
 emp.setGender(gender);  
 emp.setRole(role);  
 return emp;  
 }  
  
 private static String getEmployeeDetails(Element element, String property) {  
 String value = element.getElementsByTagName(property).item(0).getTextContent();  
 return value;  
 }  
  
}

Класс SearchXML:

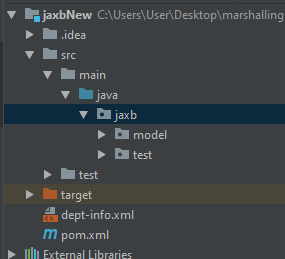
package com.DOMParse.Service;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.Scanner;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
  
import com.DOMParse.Entity.Employee;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
import org.xml.sax.SAXException;  
  
public class SearchXML {  
  
 public static void main(String[] args) throws ParserConfigurationException,SAXException, IOException {  
  
  
 File file = new File("src/main/resources/employee.xml");  
 Scanner input = null;  
 try {  
 input = new Scanner(System.*in*);  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(file);  
 System.*out*.println("Enter employee id : ");  
 int id = input.nextInt();  
 Employee employee = *getEmployee*(document, id);  
 if (employee == null) {  
 System.*out*.println("Employee not exist with id = " + id);  
 } else {  
 System.*out*.println(employee);  
 }  
  
 } finally {  
 if (input != null) {  
 input.close();  
 }  
 }  
  
 }  
  
 private static Employee getEmployee(Document document, int id) {  
  
 NodeList list = document.getElementsByTagName("Employee");  
 int length = list.getLength();  
 Employee emp = null;  
 String name = null, gender = null, role = null;  
 int empId = 0, age = 0;  
 for (int i = 0; i < length; i++) {  
 Node node = list.item(i);  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 if (String.*valueOf*(id).equals(element.getAttribute("id"))) {  
 emp = new Employee();  
 empId = Integer.*parseInt*(element.getAttribute("id"));  
 emp.setId(empId);  
 name = element.getElementsByTagName("name").item(0).getTextContent();  
 emp.setName(name);  
 age = Integer.*parseInt*(element.getElementsByTagName("age").item(0).getTextContent());  
 emp.setAge(age);  
 gender = element.getElementsByTagName("gender").item(0).getTextContent();  
 emp.setGender(gender);  
 role = element.getElementsByTagName("role").item(0).getTextContent();  
 emp.setRole(role);  
 }  
 }  
 }  
  
 return emp;  
 }  
  
}

И, наконец, класс WriteXML для записи общей информации:

package com.DOMParse;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.transform.OutputKeys;  
import javax.xml.transform.Transformer;  
import javax.xml.transform.TransformerFactory;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
  
import com.DOMParse.Entity.Employee;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
  
public class WriteXML {  
  
 public static void main(String[] args) throws IOException {  
  
 BufferedReader input = null;  
  
 try {  
 input = new BufferedReader(new InputStreamReader(System.*in*));  
 File file = new File("src/main/resources/employee.xml");  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.newDocument();  
 Element root = document.createElement("Employees");  
 document.appendChild(root);  
 System.*out*.println("Enter how many employess data you want to add? : ");  
 int count = Integer.*parseInt*(input.readLine());  
 String name = null, gender = null, role = null;  
 int age = 0;  
 Element element = null;  
 for (int i = 1; i <= count; i++) {  
 System.*out*.println("\*\*\*\*\*\*\* Employee : " + i + " \*\*\*\*\*\*\*\*");  
 System.*out*.println("Enter name : ");  
 name = input.readLine();  
 System.*out*.println("Enter gender : (male/female)");  
 gender = input.readLine();  
 System.*out*.println("Enter age : ");  
 age = Integer.*parseInt*(input.readLine());  
 System.*out*.println("Enter role : ");  
 role = input.readLine();  
 Employee employee = new Employee();  
 employee.setName(name);  
 employee.setGender(gender);  
 employee.setAge(age);  
 employee.setRole(role);  
 element = *getEmployeeNode*(employee, i, document);  
 root.appendChild(element);  
 }  
  
 TransformerFactory tFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = tFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = new DOMSource(document);  
 StreamResult result = new StreamResult(file);  
 transformer.transform(source, result);  
 System.*out*.println("Employee data has been added successfully.");  
  
 } catch (Exception exception) {  
 exception.printStackTrace();  
 } finally {  
 if (input != null) {  
 input.close();  
 }  
 }  
  
 }  
  
 private static Element getEmployeeNode(Employee employee, int id,Document document) {  
  
 Element element = document.createElement("Employee");  
 element.setAttribute("id", String.*valueOf*(id));  
 Element name = *getPropertyNode*("name", document, employee.getName());  
 element.appendChild(name);  
 Element gender = *getPropertyNode*("gender", document,employee.getGender());  
 element.appendChild(gender);  
 Element age = *getPropertyNode*("age", document,String.*valueOf*(employee.getAge()));  
 element.appendChild(age);  
 Element role = *getPropertyNode*("role", document, employee.getRole());  
 element.appendChild(role);  
 return element;  
 }  
  
 private static Element getPropertyNode(String property, Document document,String value) {  
  
 Element element = document.createElement(property);  
 element.setTextContent(value);  
 return element;  
 }  
  
}

1. **JAXB** парсер.

Создадим проект с применением JAXB парсера.

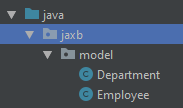


Начнем с xml файла и структуры записи. В примере ниже файл будет содержать записи, однако, вы можете оставить его пустым, в любом случае данные поместим через Java код.

Файл dept-info.xml:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<department>  
 <deptNo>D01</deptNo>  
 <deptName>ACCOUNTING</deptName>  
 <location>NEW YORK</location>  
 <employees>  
 <employee>  
 <empNo>E01</empNo>  
 <empName>Tom</empName>  
 </employee>  
 <employee>  
 <empNo>E02</empNo>  
 <empName>Mary</empName>  
 <managerNo>E01</managerNo>  
 </employee>  
 <employee>  
 <empNo>E03</empNo>  
 <empName>John</empName>  
 </employee>  
 </employees>  
</department>

В пакете model создадим классы



Данные классы будут объектным отображением нашей структуры xml документа. Т.к. наш парсер является объектным, мы обязаны четко распределить роли узлов в контексте описания документа.

Рассмотрим детально наши классы:

package jaxb.model;  
  
*/\*\*  
 \* Created by IntelliJ IDEA.<br/>  
 \* User: User  
 \* Date: 17.01.2022  
 \* Time: 19:42  
 \*/*import java.util.List;  
  
import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlElementWrapper;  
import javax.xml.bind.annotation.XmlRootElement;  
  
@XmlRootElement(name = "department")  
@XmlAccessorType(XmlAccessType.*FIELD*)  
public class Department {  
  
 private String deptNo;  
 private String deptName;  
 private String location;  
  
 @XmlElementWrapper(name = "employees")  
 @XmlElement(name = "employee")  
 private List<Employee> employees;  
  
 */\*\*  
 \* This default constructor is required if there are other constructors.  
 \*/* public Department() {  
  
 }  
  
 public Department(String deptNo, String deptName, String location) {  
 this.deptNo = deptNo;  
 this.deptName = deptName;  
 this.location = location;  
 }  
  
 public String getDeptNo() {  
 return deptNo;  
 }  
  
 public void setDeptNo(String deptNo) {  
 this.deptNo = deptNo;  
 }  
  
 public String getDeptName() {  
 return deptName;  
 }  
  
 public void setDeptName(String deptName) {  
 this.deptName = deptName;  
 }  
  
 public String getLocation() {  
 return location;  
 }  
  
 public void setLocation(String location) {  
 this.location = location;  
 }  
  
 public List<Employee> getEmployees() {  
 return employees;  
 }  
  
 public void setEmployees(List<Employee> employees) {  
 this.employees = employees;  
 }  
  
}

Обратите внимание на Аннотации. XmlRootElement будет указывать на наш корневой узел структуры (в индивидуальном задании обратите внимание на данный момент при выполнении ДЗ, не ошибитесь с корневыми элементами).

XmlAccessorType используется для указания метода доступа к атрибутам объектов Java при создании файлов XML из объектов Java. Часто используется с @XmlRootElement, @XmlType. Его значения атрибута - это 4 значения перечисления XmlAccessType, которые делятся на:

XmlAccessType.FIELD: все переменные-члены в объекте java

XmlAccessType.PROPERTY: все переменные-члены в объекте java, доступ к которым получают getter / setter

XmlAccessType.PUBLIC\_MEMBER: все переменные-члены публичного доступа в объекте java и переменные-члены, к которым обращается getter / setter

XmlAccessType.NONE: все атрибуты Java-объекта не сопоставлены с элементами XML

После чего идут поля, которые символизируют содержание нашего корневого элемента.

Не забываем объявить коллекцию сотрудников – т.к. их может быть у нас множество.

XmlElementWrapper указывает на оболочку, которая будет оборачивать данное содержимое. Однако, сотрудников может быть множество, поэтому мы обязаны указать тег для определения конкретного из них. Для этого используем уже аннотацию XmlElement.

После чего наш класс будет содержать Конструкторы и Геттеры Сеттеры по стандартным правилам POJO класса.

Отлично. Первый шаг готов, теперь нам необходима структура самого сотрудника. Иными словами, мы создали вертикаль – Департамент, который описывает объект конкретного департамента и ссылается на сотрудников.

Добавим POJO класс для сотрудника. Обратите внимание на аннотации:

package jaxb.model;  
  
*/\*\*  
 \* Created by IntelliJ IDEA.<br/>  
 \* User: User  
 \* Date: 17.01.2022  
 \* Time: 19:41  
 \*/*import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlRootElement;  
  
@XmlRootElement(name = "employee")  
@XmlAccessorType(XmlAccessType.*FIELD*)  
public class Employee {  
  
 private String empNo;  
 private String empName;  
 private String managerNo;  
  
 */\*\*  
 \* This default constructor is required if there are other constructors.  
 \*/* public Employee() {  
  
 }  
  
 public Employee(String empNo, String empName, String managerNo) {  
 this.empNo = empNo;  
 this.empName = empName;  
 this.managerNo = managerNo;  
 }  
  
 public String getEmpNo() {  
 return empNo;  
 }  
  
 public void setEmpNo(String empNo) {  
 this.empNo = empNo;  
 }  
  
 public String getEmpName() {  
 return empName;  
 }  
  
 public void setEmpName(String empName) {  
 this.empName = empName;  
 }  
  
 public String getManagerNo() {  
 return managerNo;  
 }  
  
 public void setManager(String managerNo) {  
 this.managerNo = managerNo;  
 }  
  
}

Как видим из данного класса – нам необходимо лишь указать корневой тег сотрудника. Привязка к данному классу идет из Департамента.

Маппинг завершен. Осталось создать наши объекты и успешно их записать в java объекты, а после маршелизовать внутрь xml документа.

Для этого создадим еще один класс:



И добавим немного кода : 😊

package jaxb.test;  
  
*/\*\*  
 \* Created by IntelliJ IDEA.<br/>  
 \* User: User  
 \* Date: 17.01.2022  
 \* Time: 19:43  
 \*/*import jaxb.model.Department;  
import jaxb.model.Employee;  
  
import javax.xml.bind.JAXBContext;  
import javax.xml.bind.Marshaller;  
import javax.xml.bind.Unmarshaller;  
import java.io.File;  
import java.io.FileReader;  
import java.util.ArrayList;  
import java.util.List;  
  
public class TestExample {  
  
 private static final String *XML\_FILE* = "dept-info.xml";  
  
 public static void main(String[] args) {  
  
 Employee emp1 = new Employee("E01", "Tom", null);  
 Employee emp2 = new Employee("E02", "Mary", "E01");  
 Employee emp3 = new Employee("E03", "John", null);  
  
 List<Employee> list = new ArrayList<Employee>();  
 list.add(emp1);  
 list.add(emp2);  
 list.add(emp3);  
  
 Department dept = new Department("D01", "ACCOUNTING", "NEW YORK");  
 List<Department> list1 = new ArrayList<Department>();  
 list1.add(dept);  
  
 dept.setEmployees(list);

Данная часть отвечает за наполнение нашего объекта. Теперь создадим маршелер и закинем все в xml:

try {  
 // create JAXB context and instantiate marshaller  
 JAXBContext context = JAXBContext.*newInstance*(Department.class);  
  
 // (1) Marshaller : Java Object to XML content.  
 Marshaller m = context.createMarshaller();  
 m.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, Boolean.*TRUE*);  
  
 m.marshal(dept, System.*out*);  
  
 // Write to File  
 File outFile = new File(*XML\_FILE*);  
 m.marshal(dept, outFile);  
  
 System.*err*.println("Write to file: " + outFile.getAbsolutePath());

И создадим анмаршаллер для извлечения данных

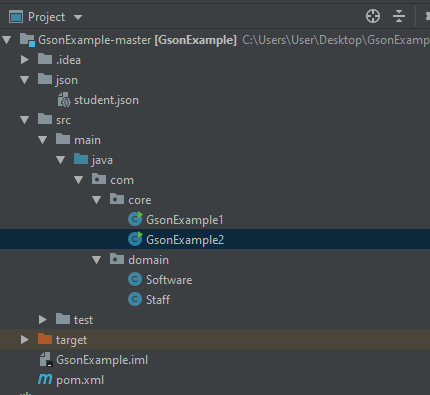
// (2) Unmarshaller : Read XML content to Java Object.  
 Unmarshaller um = context.createUnmarshaller();  
  
 // XML file create before.  
  
  
 Department deptFromFile1 = (Department) um.unmarshal(new FileReader(  
 *XML\_FILE*));  
 List<Employee> emps = deptFromFile1.getEmployees();  
 for (Employee emp : emps) {  
 System.*out*.println("Employee: " + emp.getEmpName());  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Готово. Наши данные можно передавать в виде XML.

**Задание:** Добавить Корневой элемент Организации и создать перечень департаментов со своими уникальными сотрудниками.

1. **Gson** паресер.

Нам понадобится следующий проект:



Рассмотрим детальнее классы.

Класс Stuff:

package com.domain;  
  
import java.math.BigDecimal;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
  
  
public class Staff {  
  
 private String name;  
 private int age;  
 private String[] position;  
 private List<String> skills;  
 private Map<String, BigDecimal> salary;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public String[] getPosition() {  
 return position;  
 }  
  
 public void setPosition(String[] position) {  
 this.position = position;  
 }  
  
 public List<String> getSkills() {  
 return skills;  
 }  
  
 public void setSkills(List<String> skills) {  
 this.skills = skills;  
 }  
  
 public Map<String, BigDecimal> getSalary() {  
 return salary;  
 }  
  
 @Override  
 public String toString() {  
 return "Staff{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 ", position=" + Arrays.*toString*(position) +  
 ", skills=" + skills +  
 ", salary=" + salary +  
 '}';  
 }  
  
 public void setSalary(Map<String, BigDecimal> salary) {  
 this.salary = salary;  
 }  
  
  
}

Наш POJO класс для представления объекта.

И еще один:

package com.domain;  
  
import java.util.List;  
  
  
public class Software {  
 List<Staff> staff;  
 @Override  
 public String toString() {  
 return "Software [employees=" + staff + "]";  
 }  
}

Класс GsonExample1

package com.core;  
  
import com.domain.Staff;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.lang.reflect.Modifier;  
import java.math.BigDecimal;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
  
public class GsonExample1 {  
  
 public static void main(String[] args) {  
  
 // pretty print  
 Gson gson = new GsonBuilder()  
 .setPrettyPrinting()  
 .excludeFieldsWithModifiers(Modifier.*TRANSIENT*, Modifier.*TRANSIENT*)  
 .create();  
  
 ArrayList<Staff> staffs = *createStaffObject*();  
  
 // Java objects to String  
 String json = gson.toJson(staffs);  
  
 //System.out.println(json);  
  
 // Java objects to File  
 try (FileWriter writer = new FileWriter("json/student.json")) {  
 gson.toJson(staffs, writer);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 private static ArrayList<Staff> createStaffObject() {  
  
 Staff staff = new Staff();  
 Staff staff1 = new Staff();  
 ArrayList<Staff> arr = new ArrayList<Staff>();  
  
  
 staff.setName("Oleg");  
 staff.setAge(35);  
 staff.setPosition(new String[]{"Founder", "CEO", "coder"});  
 Map<String, BigDecimal> salary = new HashMap() {{  
 put("2010", new BigDecimal(10000));  
 put("2012", new BigDecimal(12000));  
 put("2018", new BigDecimal(14000));  
 }};  
 staff.setSalary(salary);  
 staff.setSkills(Arrays.*asList*("java", "python", "node", "kotlin"));  
  
  
  
  
 staff1.setName("Oleg1");  
 staff1.setAge(351);  
 staff1.setPosition(new String[]{"Founder1", "CEO1", "coder1"});  
 Map<String, BigDecimal> salary1 = new HashMap() {{  
 put("2010", new BigDecimal(100001));  
 put("2012", new BigDecimal(120001));  
 put("2018", new BigDecimal(140001));  
 }};  
 staff1.setSalary(salary);  
 staff1.setSkills(Arrays.*asList*("java1", "python1", "node1", "kotlin1"));  
  
 arr.add(staff);  
 arr.add(staff1);  
  
 return arr;  
  
 }  
  
}

Данный класс необходим для создания маршалинга, сздания объекта и записи его в файл.

Класс GsonExample2

package com.core;  
  
import com.domain.Software;  
import com.domain.Staff;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.reflect.TypeToken;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.io.Reader;  
import java.lang.reflect.Modifier;  
import java.lang.reflect.Type;  
import java.util.List;  
  
public class GsonExample2 {  
  
 public static void main(String[] args) {  
  
 Gson gson = new GsonBuilder()  
 .setPrettyPrinting()  
 .excludeFieldsWithModifiers(Modifier.*TRANSIENT*, Modifier.*TRANSIENT*)  
 .create();  
  
 try (Reader reader = new FileReader("json/student.json")) {  
  
 // Convert JSON File to Java Object  
 // Staff staff = gson.fromJson(reader, Staff.class);  
  
 /\* Software software = gson.fromJson(reader, Software.class);  
 System.out.println(software);\*/  
  
  
 Type collectionType = new TypeToken<List<Staff>>(){}.getType();  
 List<Staff> lcs = (List<Staff>) new Gson()  
 .fromJson( reader , collectionType);  
  
 // print staff  
 System.*out*.println(lcs);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
}

Данный класс позволит прочитать файл json.

**Задание: Добавьте еще одного сотрудника и прочтите данные из файла.**

Далее поработаем с сериализацией. Если нам не нужны части данных, то мы можем их игнорировать:

Gson gson = new GsonBuilder()  
 .setPrettyPrinting()  
 .excludeFieldsWithModifiers(Modifier.*STATIC*, Modifier.*TRANSIENT*)  
 .create();

Обратите внимание на классы рефлексии!

После чего можете смело добавлять атрибуты transient в pojo. Данные не будут затронуты. Так же можно настраивать любые данные.

Существует возможность применения аннотаций:

Gson gson = new GsonBuilder()  
 .setPrettyPrinting()  
 .excludeFieldsWithoutExposeAnnotation()  
 .create();

После чего добавим в pojo

private String name;  
private int age;  
@Expose(serialize = true, deserialize = true)  
private String[] position;  
private List<String> skills;  
private Map<String, BigDecimal> salary;

Обратите внимание на данные в файле.

Поработаем с рефлексией

Создайте интерфейс:

package com.core;  
  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target({ElementType.*FIELD*})  
public @interface ExcludeField {  
}

Создадим класс правил игнорирования:

package com.core;  
  
import com.google.gson.ExclusionStrategy;  
import com.google.gson.FieldAttributes;  
  
public class CustomExclusionStrategy implements ExclusionStrategy {  
  
 private final Class<?> typeToSkip;  
  
 public CustomExclusionStrategy(Class<?> typeToSkip) {  
 this.typeToSkip = typeToSkip;  
 }  
  
 @Override  
 public boolean shouldSkipField(FieldAttributes f) {  
  
 // if field name 'salary`, skip  
 if ("salary".equals(f.getName())) {  
 return true;  
 }  
  
 // if found @ExcludeField, skip  
 if (f.getAnnotation(ExcludeField.class) != null) {  
 return true;  
 }  
  
 return false;  
 }  
  
 @Override  
 public boolean shouldSkipClass(Class<?> clazz) {  
 return (clazz == typeToSkip);  
 }  
  
}

Маркируем pojo

private String name;  
private int age;  
@ExcludeField  
private String[] position;  
private List<String> skills;  
private Map<String, BigDecimal> salary;

Добавляем опцию:

Gson gson = new GsonBuilder()  
 .setExclusionStrategies(new CustomExclusionStrategy(List.class)) // exclude all List fields.  
 .create();

Запускаем, проверяем.