JAXB exmaple: Marshalling and Unmarshalling list or set of objects

We know that [JAXB(Java Architecture for XML Binding)](http://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. JAXB mostly is used while implementing webservices or any other such client interface for an application where data needs to be transferred in XML format instead of HTML format which is default in case of visual client like web browsers.

A good example is facebook APIs. Facebook has exposed its services through some open endpoints in form of **[RESTful webservices](http://howtodoinjava.com/restful-web-service/" \o "resteasy tutorials)** where you hit a URL and post some parameters, and API return you the data in xml format. Now it is upto you, how you use that data.

In this post, I am giving an example of marshalling and unmarshalling of collection of objects. These [**collections in java**](http://howtodoinjava.com/category/core-java/collections/) can be of type : List and Set implementations e.g. ArrayList or HashSet.

**Maven dependencies**

To run JAXB examples, we need to add run time [**maven**](http://howtodoinjava.com/maven/) dependencies like below:

|  |
| --- |
| <dependency>      <groupId>com.sun.xml.bind</groupId>      <artifactId>jaxb-core</artifactId>      <version>2.2.8-b01</version>  </dependency>  <dependency>      <groupId>com.sun.xml.bind</groupId>      <artifactId>jaxb-impl</artifactId>      <version>2.2-promoted-b65</version>  </dependency> |

**Model classes**

I have created a model class “Employee.java” which has some common fields. I want to build code which could parse a set of employees. Please note that JAXB requires [**@XmlRootElement**](http://docs.oracle.com/javaee/5/api/javax/xml/bind/annotation/XmlRootElement.html) annotation on top most class which we are going to marshal or unmarshal. ArrayList class is part of collection framework and it does not have any JAXB annotations. So We need to have another class “Employees.java” which will represent the set of employees. Now in this class we can add any annotation we like.

**Employee.java**

|  |
| --- |
| @XmlRootElement(name = "employee")  @XmlAccessorType (XmlAccessType.FIELD)  public class Employee  {      private Integer id;      private String firstName;      private String lastName;      private double income;        //Getters and Setters  } |

**Employees.java**

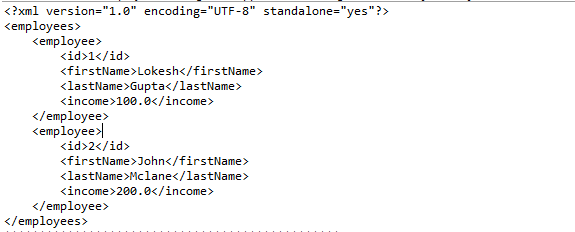
|  |
| --- |
| import java.util.List;    import javax.xml.bind.annotation.XmlAccessType;  import javax.xml.bind.annotation.XmlAccessorType;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "employees")  @XmlAccessorType (XmlAccessType.FIELD)  public class Employees  {      @XmlElement(name = "employee")      private List<Employee> employees = null;        public List<Employee> getEmployees() {          return employees;      }        public void setEmployees(List<Employee> employees) {          this.employees = employees;      }  } |

**Marshalling example**

Marshalling is “to convert the java object into xml representation”. In below example code, I am writing the list of employees first in console, and then in a file.

|  |
| --- |
| //Initialize the employees list  static Employees employees = new Employees();  static  {      employees.setEmployees(new ArrayList<Employee>());      //Create two employees      Employee emp1 = new Employee();      emp1.setId(1);      emp1.setFirstName("Lokesh");      emp1.setLastName("Gupta");      emp1.setIncome(100.0);        Employee emp2 = new Employee();      emp2.setId(2);      emp2.setFirstName("John");      emp2.setLastName("Mclane");      emp2.setIncome(200.0);        //Add the employees in list      employees.getEmployees().add(emp1);      employees.getEmployees().add(emp2);  }  private static void marshalingExample() throws JAXBException  {      JAXBContext jaxbContext = JAXBContext.newInstance(Employees.class);      Marshaller jaxbMarshaller = jaxbContext.createMarshaller();        jaxbMarshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);        //Marshal the employees list in console      jaxbMarshaller.marshal(employees, System.out);        //Marshal the employees list in file      jaxbMarshaller.marshal(employees, new File("c:/temp/employees.xml"));  } |

Output of above code is :



*JAXB marshalling example output*

**Unmarshalling example**

Unmarshalling is the process of converting the xml back to java object. Let’s see the example of our Employees class.

|  |
| --- |
| private static void unMarshalingExample() throws JAXBException  {      JAXBContext jaxbContext = JAXBContext.newInstance(Employees.class);      Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();        //We had written this file in marshalling example      Employees emps = (Employees) jaxbUnmarshaller.unmarshal( new File("c:/temp/employees.xml") );        for(Employee emp : emps.getEmployees())      {          System.out.println(emp.getId());          System.out.println(emp.getFirstName());      }  }    Output:    1  Lokesh  2  John |