# https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/

# Использование аннотаций с JAXB, генерация XML schema из java-кода

### [Галерея](https://dev64.wordpress.com/type/gallery/)

Posted on [Май 15, 2012](https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/)

Однажды, возникла необходимость настройки формата дат внутри XML документов. Работа с документами осуществлялась с помощью JAXB. Всвязи с этим коллеги нашли в интернете замечательную публикацию на сайте <http://ooxs-be.goracer.nl/EN/java/Java%20and%20XML%20Binding.html>

Ниже перевод не совсем близкий к тексту. Я слегка видоизменяю его для улучшения понимаемости. (По крайней мере мне хочется её улучшить). Если что-то не совсем понятно, смотрите оригинал.

**Introduction**

При работе с XML в Java, он может рассматриваться просто как текст, который должен быть прочитан изменен и куда-то записан. Т.о. это может быть сделано обычной конкатенацией строк или с помощью DOM-модели в памяти для доступа, создания или модификации документа.

В большинстве случаев, однако, кроме XML документа имеется набор классов — модель в виде beans или POJO, не важно как классы представлены и называются, но эти классы должны преобразовываться в XML документ и обратно из XML документа превращаться в набор классов. Это нужно для посылки этих классов в запросе на веб-сервис, для сохранения в файле и т.п.

**Такая взаимосвязь модели Java объектов и XML представления документов называется англоговорящими программистами binding. Создание объектов из XML называется unmarshalling, а обратная операция генерации XML из Java объектов — marshalling**.

Есть различные способы для реализации unmarshalling-а и marshalling-a:

* Написать SAX парсер для создания Java объектов
* Работать с DOM деревом для извлечения нужных объектов из него
* Использовать специальный API такой как JAXB, Apache Commons Betwixt, и т.д.

**Java Architecture for XML Binding (JAXB)**

JAXB не только API. Существуют стандартные средства для генерации Java классов из [XML schema](https://dev64.wordpress.com/category/xml/xml-schema-xml/) (to form binding to XML), и для генерации [XML schema](https://dev64.wordpress.com/category/xml/xml-schema-xml/) из Java классов с расставленными аннотациями. Эти средства появились в Java как часть пакета JAXB.

**Практический пример Java классов с JAXB аннотациями**

Подходящий случай для использования JAXB — когда у вас имеется набор классов в приложении, которые вы хотите писать и читать в XML формате. Мы используем очень простой пример моделирования группы людей. У нас имеются два класса Group и People которые должны преобразовываться в XML и обратно. Дополнительно есть класс Application который и будет производить эти операции.

Преимущество JAXB в том, что в Java 6 мы используем стандартный API, ничего дополнительно не требуется устанавливать, конфигурировать и т.д.

Ниже корневой класс нашего XML документа, мы должны добавить аннотацию @XmlRootElement. propOrder = { «name», «members» } указывает в каком порядке элементы будут записываться в XML теги.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | package be.ooxs.example.jaxb;    import java.util.ArrayList;  import java.util.List;    import javax.xml.bind.annotation.XmlRootElement;  import javax.xml.bind.annotation.XmlType;    @XmlType(propOrder = { "name", "members" }, name = "group")  @XmlRootElement  public class Group {      private String name;      private List<Person> members = new ArrayList<Person>();        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public List<Person> getMembers() {          return members;      }        public void setMembers(List<Person> members) {          this.members = members;      }  } |

Подобным образом добавляем аннотации в дочерний класс

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package be.ooxs.example.jaxb;    import java.util.Date;    import javax.xml.bind.annotation.XmlType;    @XmlType(propOrder = { "lastName", "firstName", "birthDate" }, name = "person")  public class Person {      private Date birthDate;      private String firstName;      private String lastName;        public Date getBirthDate() {          return birthDate;      }        public void setBirthDate(Date birthDate) {          this.birthDate = birthDate;      }        public String getFirstName() {          return firstName;      }        public void setFirstName(String firstName) {          this.firstName = firstName;      }        public String getLastName() {          return lastName;      }        public void setLastName(String lastName) {          this.lastName = lastName;      }    } |

И, наконец, ниже класс Application выполняющий сериализацию (marshalling-unmarshalling) вышеприведенных классов с помощью JAXB:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | package be.ooxs.example.jaxb;    import java.io.ByteArrayInputStream;  import java.io.ByteArrayOutputStream;  import java.io.InputStream;  import java.text.ParseException;  import java.text.SimpleDateFormat;  import java.util.logging.Level;  import java.util.logging.Logger;    import javax.xml.bind.JAXBContext;  import javax.xml.bind.JAXBException;  import javax.xml.bind.Marshaller;  import javax.xml.bind.Unmarshaller;    public class Application {    private SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");      public Group createJavaObjectExample1() {     Group group = new Group();     group.setName("Test Group 1");     try {      group.getMembers().add(createPerson("Alice", "Anderssen", "1970-01-01"));      group.getMembers().add(createPerson("Bert", "Bobo", "1980-02-02"));     } catch (ParseException exception) {      Logger.getLogger(Application.class.getName()).         log(Level.ALL, "createJavaObjectExample1 threw ParseException", exception);     }     return group;    }       public Person createPerson(String firstName, String lastName, String birthDate) throws ParseException {       Person person = new Person();       person.setBirthDate(format.parse(birthDate));       person.setFirstName(firstName);       person.setLastName(lastName);       return person;     }       public void marshallExample() {       //...     }       public void unmarshallExample() {      //...     }       public static void main(String[] args) {      Application instance = new Application();      instance.marshallExample();      instance.unmarshallExample();      System.out.println("Done");     }  } |

Добавим код для генерации XML, и выполним приложение добавив в группу пару элементов для демонстрации JAXB API:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public void marshallExample() {    try {     JAXBContext context = JAXBContext.newInstance(Group.class);     Marshaller marshaller = context.createMarshaller();     marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);     marshaller.marshal(createJavaObjectExample1(), System.out);    } catch (JAXBException exception) {     Logger.getLogger(Application.class.getName()).        log(Level.SEVERE, "marshallExample threw JAXBException", exception);    }  } |

Благодаря аннотациям, лишь три строчки кода генерируют XML. (4 для лучшего форматирования для улучшения читаемости)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <group>      <name>Test Group 1</name>      <members>          <lastName>Anderssen</lastName>          <firstName>Alice</firstName>          <birthDate>1970-01-01T00:00:00+01:00</birthDate>      </members>      <members>          <lastName>Bobo</lastName>          <firstName>Bert</firstName>          <birthDate>1980-02-02T00:00:00+01:00</birthDate>      </members>  </group> |

Однако в выходном XML есть ряд неожиданных элементов.  
Каждый элемент Person помещён внутрь тега members вместо тега person. Это мы изменим позже. А пока посмотрим пример обратного преобразования. Сначала мы повторим предыдущий пример, создадим XML, потом запишем его в byte array. Затем сделаем unmarshalling из сгенерированного предварительно XML.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | public void unmarshallExample() {      try {          JAXBContext context =             JAXBContext.newInstance(Group.class);          Marshaller marshaller =             context.createMarshaller();          marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT,             Boolean.TRUE);            //write XML to an array of bytes          ByteArrayOutputStream baos =              new ByteArrayOutputStream();          marshaller.marshal(createJavaObjectExample1(), baos);            //read XML from array of bytes          InputStream bais =             new ByteArrayInputStream(baos.toByteArray());          Unmarshaller unmarshaller =             context.createUnmarshaller();          Object o = unmarshaller.unmarshal(bais);            //prove the Group is recreated          Group groupCopy = (Group) o;          System.out.println("Objects created from XML:");          System.out.println(groupCopy.getName());          for (Person person : groupCopy.getMembers()) {              System.out.println(person.getFirstName());          }      } catch (JAXBException exception) {          Logger.getLogger(Application.class.getName()).            log(Level.SEVERE, "marshallExample threw JAXBException", exception);      }  } |

В результате теста дополнительно к XML получаем следующий output:

Objects created from XML:

Test Group 1

Alice

Bert

Done

JAXB утилиты входят в стандартную инсталляцию Java. При установке Java 6 JDK они устанавливаются в каталог $JAVA\_HOME/bin.  
Например в дистрибутивах Linux подобных Ubuntu, они будут по умолчанию доступны для вызова из командной строки.

**Генерация XML schema из Java-классов.**

Чтобы JAXB утилиты стали доступны под Windows, нужно в переменных окружения указать путь к поддиректорию bin в каталоге с JDK. Для генерации [XML schema](https://dev64.wordpress.com/2012/02/02/xml-schema/) из Java кода с добавленными аннотациями, используется утилита schemagen:

Допустим мы находимся в корневом директории исходников. Т.е. в нём у нас подкаталог be/ooxs/example/jaxb/ с исходниками:  
Group.java, Person.java, Application.java

Тогда находясь в директории с исходниками мы генерируем [XML схему](https://dev64.wordpress.com/2012/02/02/xml-schema/) такой командой (я тестируюсь под Windows):

|  |  |
| --- | --- |
| 1 | > schemagen -d . -cp . be/ooxs/example/jaxb/Group.java |

Здесь

* -d . задает, что [схема](https://dev64.wordpress.com/category/xml/xml-schema-xml/) будет положена в текущий директорий
* -cp . — classpath. Корневой директорий наших исходников. Т.к. мы в нём и находимся. То указываем точку.
* третий параметр — путь к .java — файлу с исходным кодом корневого элемента нашего XML документа.

В итоге будет получен [схема файл](https://dev64.wordpress.com/category/xml/xml-schema-xml/), который вы можете использовать по мере необходимости. Заметьте, что порядок элементов lastName, firstName, birthDate как определено в аннотациях.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <xs:schema version="1.0" xmlns:xs="<http://www.w3.org/2001/XMLSchema>">      <xs:element name="group" type="group"/>      <xs:complexType name="group">      <xs:sequence>        <xs:element name="name" type="xs:string" minOccurs="0"/>        <xs:element name="members" type="person" nillable="true" minOccurs="0" maxOccurs="unbounded"/>      </xs:sequence>    </xs:complexType>      <xs:complexType name="person">      <xs:sequence>        <xs:element name="lastName" type="xs:string" minOccurs="0"/>        <xs:element name="firstName" type="xs:string" minOccurs="0"/>        <xs:element name="birthDate" type="xs:dateTime" minOccurs="0"/>      </xs:sequence>    </xs:complexType>  </xs:schema> |

**Модифицируем аннотации**

Для получения слегка улучшенной XML структуры, добавим несколько вещей:

Для начала добавим пару аннотаций, первая группирует все дочерние элементы используя дополнительный members элемент, вторая изменяет имя каждого дочернего элемента на person.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @XmlElementWrapper(name = "members")  @XmlElement(name = "person")  public List<Person> getMembers() {    return members;  } |

В результате при повторении теста получим XML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <group>      <name>Test Group 1</name>      <members>          <person>              <lastName>Anderssen</lastName>              <firstName>Alice</firstName>              <birthDate>1970-01-01T00:00:00+03:00</birthDate>          </person>          <person>              <lastName>Bobo</lastName>              <firstName>Bert</firstName>              <birthDate>1980-02-02T00:00:00+03:00</birthDate>          </person>      </members>  </group> |

Перегенерируем [XML schema](https://dev64.wordpress.com/2012/02/02/xml-schema/):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <xs:schema version="1.0" xmlns:xs="<http://www.w3.org/2001/XMLSchema>">      <xs:element name="group" type="group"/>      <xs:complexType name="group">      <xs:sequence>        <xs:element name="name" type="xs:string" minOccurs="0"/>        <xs:element name="members" minOccurs="0">          <xs:complexType>            <xs:sequence>              <xs:element name="person" type="person" minOccurs="0" maxOccurs="unbounded"/>            </xs:sequence>          </xs:complexType>        </xs:element>      </xs:sequence>    </xs:complexType>      <xs:complexType name="person">      <xs:sequence>        <xs:element name="lastName" type="xs:string" minOccurs="0"/>        <xs:element name="firstName" type="xs:string" minOccurs="0"/>        <xs:element name="birthDate" type="xs:dateTime" minOccurs="0"/>      </xs:sequence>    </xs:complexType>  </xs:schema> |

[Продолжение здесь…](https://dev64.wordpress.com/2012/05/15/extended-jaxb-annotations/)

# Расширенный пример использования JAXB аннотаций для форматирования даты в XML

[В предыдущем посте](https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/) были описаны эксперименты с JAXB аннотациями. Этот пост — продолжение начатых экспериментов. Первичный источник информации:

В классы описанные [в предыдущем посте](https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/) добавляем дополнительные enum поля. Эти поля описываем с помощью аннотаций, чтобы получить нужный нам вид XML.

Для начала создаем enum PersonType:

|  |  |
| --- | --- |
| 1  2  3  4  5 | package be.ooxs.example.jaxb.extended;    public enum PersonType {      CUSTOMER, EMPLOYEE, MANAGER  } |

enum Status:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package be.ooxs.example.jaxb.extended;    import javax.xml.bind.annotation.XmlEnumValue;    public enum Status {      @XmlEnumValue(value = "pending")      PENDING\_APPROVAL,      @XmlEnumValue(value = "active")      ACTIVE,      @XmlEnumValue(value = "inactive")      INACTIVE  } |

Здесь добавлены аннотации задающие соответствие enum значений и соответствующих значений в xml. PENDING\_APPROVAL-> pending

Ещё, так как мы занимаемся украшательством, почему не улучшить формат даты. Мы хотим чтобы дата писалась не по умолчанию, а по заданному нами формату. Для реализации этого создаём класс-адаптер:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package be.ooxs.example.jaxb.extended;    import java.text.SimpleDateFormat;  import java.util.Date;    import javax.xml.bind.annotation.adapters.XmlAdapter;    public class ShortDateFormatter extends XmlAdapter<String, Date> {      SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");        public Date unmarshal(String date) throws Exception {          return formatter.parse(date);      }        public String marshal(Date date) throws Exception {          return formatter.format(date);      }  } |

Добавляем аннотацию к getBirthDate().

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ...      @XmlJavaTypeAdapter(ShortDateFormatter.class)      public Date getBirthDate() {          return birthDate;      }      ... |

Т.о. наш класс Person выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | package be.ooxs.example.jaxb.extended;    import java.util.Date;    import javax.xml.bind.annotation.XmlAttribute;  import javax.xml.bind.annotation.XmlType;    @XmlType(propOrder = { "lastName", "firstName", "birthDate", "type" }, name = "person")  public class Person {      private Date birthDate;      private String firstName;      private String lastName;      private PersonType type;      @XmlAttribute(name = "active")      private Status status = Status.PENDING\_APPROVAL;        public PersonType getType() {          return type;      }        public void setType(PersonType type) {          this.type = type;      }        public Person delete() {          status = Status.INACTIVE;          return this;      }        public Person approve() {          status = Status.ACTIVE;          return this;      }        @XmlJavaTypeAdapter(ShortDateFormatter.class)      public Date getBirthDate() {          return birthDate;      }        public void setBirthDate(Date birthDate) {          this.birthDate = birthDate;      }        public String getFirstName() {          return firstName;      }        public void setFirstName(String firstName) {          this.firstName = firstName;      }        public String getLastName() {          return lastName;      }        public void setLastName(String lastName) {          this.lastName = lastName;      }    } |

В вышеприведённом коде аннотация @XmlAttribute(name = «active») перед полем status задаёт, что значение этого поля будет значением атрибута active, а не тегом active.

Не добавляем аннотацию к полю. Если попытаться сделать это, тогда получим Exception в рантайме, поскольку и поле и геттер объявляют свойство с одинаковым именем birthDate.

Этого можно избежать с помощью аннотации @XmlAccessorType(XmlAccessType.FIELD) перед классом, но это влияет как значения будут получаться из класса для всех полей. Не лучший подход. (JAXB будет использовать поля, а не геттеры в таком случае). Ниже кусок исключения, генерируемого в данном случае. Чтобы если у вас оно возникло, вы имели представление из-за чего это может возникнуть.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | com.sun.xml.internal.bind.v2.runtime.IllegalAnnotationsException: 2 counts of IllegalAnnotationExceptions  There are two properties named "birthDate"      this problem is related to the following location:          at public java.util.Date be.ooxs.example.jaxb.extended.Person.getBirthDate()          ...      this problem is related to the following location:          at private java.util.Date be.ooxs.example.jaxb.extended.Person.birthDate |

Запускаем изменённый код на тестирование и получает такой output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <group>      <name>Test Group 1</name>      <members>          <person active="pending">              <lastName>Anderssen</lastName>              <firstName>Alice</firstName>              <birthDate>1970-01-01</birthDate>          </person>          <person active="pending">              <lastName>Bobo</lastName>              <firstName>Bert</firstName>              <birthDate>1980-02-02</birthDate>          </person>      </members>  </group>  Objects created from XML:  Test Group 1  Alice  Bert  Done |

Здесь видно, что в XML появился атрибут active=»pending», а также видно, что дата стала форматироваться без времени.

Пробую перегенерировать схему. Делаю это командой:

|  |  |
| --- | --- |
| 1 | > schemagen -d . -cp . be/ooxs/example/jaxb/Group.java |

Здесь

* -d . задает, что схема будет положена в текущий директорий
* -cp . — classpath. Корневой директорий наших исходников. Т.к. мы в нём и находимся. То указываем точку.
* третий параметр — путь к .java — файлу с исходным кодом корневого элемента нашего XML документа.

Подробности смотрите [в предыдущем посте.](https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/)

В итоге получаю схему:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <xs:schema version="1.0" xmlns:xs="<http://www.w3.org/2001/XMLSchema>">      <xs:element name="group" type="group"/>      <xs:complexType name="group">      <xs:sequence>        <xs:element name="name" type="xs:string" minOccurs="0"/>        <xs:element name="members" minOccurs="0">          <xs:complexType>            <xs:sequence>              <xs:element name="person" type="person" minOccurs="0" maxOccurs="unbounded"/>            </xs:sequence>          </xs:complexType>        </xs:element>      </xs:sequence>    </xs:complexType>      <xs:complexType name="person">      <xs:sequence>        <xs:element name="lastName" type="xs:string" minOccurs="0"/>        <xs:element name="firstName" type="xs:string" minOccurs="0"/>        <xs:element name="birthDate" type="xs:string" minOccurs="0"/>        <xs:element name="type" type="personType" minOccurs="0"/>      </xs:sequence>      <xs:attribute name="active" type="status"/>    </xs:complexType>      <xs:simpleType name="personType">      <xs:restriction base="xs:string">        <xs:enumeration value="CUSTOMER"/>        <xs:enumeration value="EMPLOYEE"/>        <xs:enumeration value="MANAGER"/>      </xs:restriction>    </xs:simpleType>      <xs:simpleType name="status">      <xs:restriction base="xs:string">        <xs:enumeration value="pending"/>        <xs:enumeration value="active"/>        <xs:enumeration value="inactive"/>      </xs:restriction>    </xs:simpleType>  </xs:schema> |

**Краткое summary**

Т.о. с помощью аннотаций к коду, решена задача форматирования даты в XML с помощью JAXB. Кроме того текущий пост и предыдущий дали пример генерирования [XML Schema](https://dev64.wordpress.com/2012/02/02/xml-schema/) из аннотированного java кода и пример маршаллинга XML.

Другие посты про технологию JAXB

* [Практический пример использования XML schema и JAXB](https://dev64.wordpress.com/2012/03/19/xml-schema-jaxb-example/)
* [Обработка JAXB исключений](https://dev64.wordpress.com/2012/04/12/jaxb-exceptions-processing/)
* [Чтение XML-документа с помощью JAXB с заменой namespace](https://dev64.wordpress.com/2012/04/26/namespace-filter-example/)
* [Использование аннотаций с JAXB, генерация XML schema из java-кода](https://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# JAXB Hello World example

With this example we shall show you how to work with the JAXB framework in Java. JAXB stands for Java Architecture for XML Binding. You can use JAXB to convert an object to XML format and write it to an XML File as well as read an XML File and construct a Java object out of it. These to operations are described in JAXB as:

Marshalling, which converts a Java object into XML format and writes it to an XML File (or generally an output stream).

Unmarshalling, which converts an XML file into a Java object.

### 1. JAXB library

As of JDK 1.6, JAXB libraries are included in the JDK. If you are using an older JDK you can download it form the [JAXB official project site](http://jaxb.java.net/).

### 2. JAXB syntax for XML Formatting

JAXB uses annotations to provide meta information that dictates the exact structure of XML elements that will be derived from the class. The annotations are quite straightforward. Of course in the [official site](http://jaxb.java.net/tutorial/) you will find all the information you need about JAXB annotations.

Here is the class that we are going to use for the demo:

Student.java:

### 3. Marshalling

Remember that Marshalling is the operation that JAXB uses in order to convert an object to XML format and write it to a File.

JXBExampleJava.java

### 4. Unmarshalling

Now we are going to do the opposite operation. That is, read an XML File and convert it into a Java Object.

JXBExampleJava.java**Output:**

First Name : Jacl Last Name : Freeman Age : 32 ID : 100

This was an example on how to work with JAXB in Java.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**JAXB marshal example**

In this example we are going to show how to use the JAXB marshal functionalities. JAXB offers the possibility to convert Java objects into XML structures and vice versa, it comes with the JRE bundle since the first versions of the JRE 1.6.

As example, we are going to create a list of museums and store it in a specific XML file, each museum contains information like its name, about permanent and special exhibitions, city where is located, etc.

The java version used for these examples is the JRE 1.8.0 for 32b but only the usage of the LocalDate class is Java 8 specific; any 1.7 version can be used. The IDE used is Eclipse SDK Version: Luna (4.4) but the code should work in any other IDE supporting java.

So, here we go…

## First of all we indicate JAXB what Java elements we want to store in our XML fileIn the code shown above, we can see three JAXB annotations:

* @XmlRootElement( name = "MUSEUM" ): indicates the root node in the xml structure, the name is the name that will appear in the xml, if no name is specified, the class name will be used.
* @XmlElement( name = "MUSEUM\_NAME" ): indicates a child node.
* @XmlAttribute( name = "children\_allowed" ): indicates an attribute of the root node.

## Next step is to marshal this object and generate the XML with the desired structure: The code is self explained and shows how a JAXB Marshaller can be used to generate an XML from a Java object. If the JAXB\_FORMATTED\_PROPERTY is set to true, this indicates JAXB to generate an XML with a proper indentation. The marshal method uses an object and an output file where to store the generated XML as parameters.

The generated XML would be:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?> | | | |
| 2 | <MUSEUM children\_allowed="false"> | |
| 3 | <MUSEUM\_NAME>Simple Museum</MUSEUM\_NAME> | | | | |
| 4 | <CITY>Oviedo, Spain</CITY> | | |
| 5 | </MUSEUM> |

We now know how to generate an XML from a Java object, now we are going to show how to work with Lists of objects

## How to store a list of museums:

It is important to notice that JAXB is not able to manage directly Lists as root element, so we need to create a new class with the list that we want to store in order to indicate JAXB what kind of XML structure it has to generate. In this example this class is called Museums and contains a List of Museum:

Until now, we just generated XML that contained elements of the type String, so we are going to see what actions are needed in order to allow JAXB to store other types that are not configured per default. In Java 8, one of the new features is the new Date API; this API offers many new possibilities and enhances the old one. One of the new classes coming with this API is the java.time.LocalDate. JAXB does not know how to manage this class, so we have to create an adapter in order to explain JAXB how to marshal and unmarshal it:

## How to store complex Java types as children nodes using an adapter: we just implement the marshal and unmarshal methods of the interface XmlAdapter with the proper types and results and afterwards, we indicate JAXB where to use it: assuming that this.from is of the type LocalDate.

The XML result would be:

|  |  |
| --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?> |
| 2 | <MUSEUM children\_allowed="false"> |
|  | <MUSEUM\_NAME>Simple Museum</MUSEUM\_NAME> |
| 4 | <CITY>Oviedo, Spain</CITY> |
| 5 | <PERMANENT\_EXHIBITION> |
| 6 | <NAME>one exhibition</NAME> |
| 7 | <FROM>2014-01-01</FROM> |
| 8 | </PERMANENT\_EXHIBITION> |
| 9 | </MUSEUM> |

Summarizing, we know how to generate XML from Java objects, we know also how to use lists within these java objects and also as root element of the XML, we saw how to adapt complex types in order that JAXB can work with them and we also made as well.  
The example bellow contains all the features explained in this article: A list of museums containing names, cities, permanent and special exhibitions with dates (using java 8 LocalDate) and list of artists in each exhibition is stored in an XML file.

## Java main program: So that is all about marshaling Java objects into XML. JAXB contains other annotations that we did not touch in this article, for an overview of all possible JAXB annotations please go to: <http://docs.oracle.com/javase/8/docs/api/javax/xml/bind/annotation/package-summary.html>

JAXB also provides the option to use XML schema definitions (XSD) in order to generate the Java classes that are going to be marshaled and unmarshaled afterwards, since this is very useful in cases like interfaces between backend systems, message queuing and others, we did not show it in this article because it would have complicated things too much.

You can find all the code used in this article with a lot of comments and explanations and some extra features in the following compressed file: [museums\_jaxb](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2014/06/museums_jaxb.zip)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# JAXB unmarshal example

In my last [article](http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-marshal-example/), I’ve explained how to marshall java objects to xml using JAXB. In this one, we are going to see how to do the complementary operation: unmarshal xml files into java objects and what should be taken into consideration while doing this operation.

For this purpose we are going to use the same example as in the [marshal example article](http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-marshal-example/). We are going to unmarshall an xml containing information about a list of museums with its main exhibitions and artists exposed.

The java version used for these examples is the JRE 1.8.0. The IDE used is Eclipse SDK Version: Luna (4.4) but the code should work in any other IDE supporting java.

The code shown bellow unmarshalls a given xml file into java objects. The classes of these objects should contain a set of attributes and annotations:

## XML file: and the Java main program: but this depends in the current Java code handling the Museums class.

The method createUnmarshaller of the class JAXBContext creates an instance of the type Unmarshaller that allows us to proceed with our tasks. If the class Museums and its members are properly configured using the right JAXB annotations and field members, everything should work fine.

The museums class contains a list of Museum items: and the Museum class contains fields that can be XML elements like the name or the city or XML attributes like the children allowance. These fields can be of any JAXB supported type:

Main annotations used

We are going to see some important points related to the configuration of the used classes and the annotations used to configure JAXB:

@XmlRootElement: This annotation binds an XML node with a class or an enum. In our example we bind, using the @XmlRootElement, the XML element <MUSEUMS>…</MUSEUMS> with the class Museums by annotating this class with @XmlRootElement( name = "MUSEUMS" ).  
For more information about this annotation please refer to [XMLRootElement](http://docs.oracle.com/javase/7/docs/api/javax/xml/bind/annotation/XmlRootElement.html)

@XmlElement: Maps XML node into a non static field of a class. In our example, among others, we map the element <MUSEUM\_NAME>Prado Museum</MUSEUM\_NAME> with the field name of the class Museum using the annotation @XmlElement( name = "MUSEUM\_NAME" ) in the setName() method.  
For more information about this one, please refer to [XMLElement](http://docs.oracle.com/javase/7/docs/api/index.html?javax/xml/bind/annotation/XmlElement.html).

@XmlAttribute: This annotation maps an XML attribute with a non static field of a class. We bind the childrenAllowed field of the Museum class with the xml attribute <MUSEUM children\_allowed=”false”> using the following code:

For more information about the annotation @XmlAttribute go to [XMLAttribute](http://docs.oracle.com/javaee/7/api/javax/xml/bind/annotation/XmlAttribute.html).

And this is how we can unmarshal an XML file into Java objects.

JAXB offers several ways to marshal and un marshal collections. In our example, we just created a Museums class that contains a list of Museum items, so JAXB can manage directly this class by simply using the annotations explained above. It is also possible to achieve something similar by using the annotations @XmlElementWrapper or @XmlList, but under my point of view, these ones are more complicated, offer less options and ties you in several ways in your class modelling.

As already mentioned in the [article](http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-marshal-example/), it is also possible to generate java classes with JAXB annotations automatically by using XSD (XML Schmema Definition). This offers also the possibility to validate the XML files against the provided XML Schema that we want to unmarshall. It is also important to mention that JAXB does not need an XSD to work (as we saw in this example), this offers Java programmers a lot of flexibility.

All the code explained in this article and some running examples can be found in the following link: [unmarshall.zip](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2014/06/unmarshall.zip)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# <http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-json-example/>

# JAXB JSON Example

In this example we shall show you how to make use of JAXB-JSON. [JAXB](https://jaxb.java.net/) is a java architecture for XML binding is an efficient technology to convert XML to and from Java Object. [EclipseLink JAXB (MOXy)](http://www.eclipse.org/eclipselink/moxy.php) is one of [JAXB](https://jaxb.java.net/) implementation which is mostly used to create java classes from XML or JSON. In Java [JAXB](https://jaxb.java.net/) provides two general purpose implementation.

* **Marshalling** – It Converts a Java object into XML or JSON.
* **Unmarshalling** – It Converts XML or JSON into a Java Object.

Now, We will demonstrate the native object-to-JSON binding MOXy JAXB introduced in EclipseLink 2.4. With MOXy as your JAXB provider you can produce/consume JSON using the standard JAXB APIs (available in Java SE 6) without adding any compile time dependencies.

## Example:

### 1. MOXy Dependency: 3. Marshal Java Object to JSON:

Create a JaxBContext using the Employee class then convert the “employee” Java object into JSON formatted string using Marshaller object with following three properties:

* MEDIA\_TYPE – Determine the produced output media type (JSON, XML).
* JSON\_INCLUDE\_ROOT – Flag to determine whether you want to include the JSON root element in the produced output or not.
* JAXB\_FORMATTED\_OUTPUT – Flag to determine whether you want to format the produced output or not.

MarshallerDemo.java: 4. Unmarshal JSON to Java Object:

Create a JaxBContext using the Employee class then read the provided JSON string and convert it back to the “employee” Java object using Unmarshaller object with following two properties:

* MEDIA\_TYPE – Determine the provided input media type (JSON, XML).
* JSON\_INCLUDE\_ROOT – Flag to determine whether you want to include the JSON root element in the provided input or not.

UnmarshallerDemo.java: Output:

|  |  |
| --- | --- |
| 1 | Employee Id: 1 |
| 2 | Employee Name: Ashraf | |

|  |  |
| --- | --- |
| 3 | Employee Skills: java,sql |

**Tip**

**Specify MOXy as the JAXB Provider (jaxb.properties)**  
To configure MOXy as your JAXB provider simply add a file named jaxb.properties in the same package as your domain model with the following entry:

|  |  |
| --- | --- |
| 1 | javax.xml.bind.context.factory=org.eclipse.persistence.jaxb.JAXBContextFactory |

## 5. Download the Source Code of this example:

This was an example of how to use JAXB-JSON to marshal and unmarshal java POJO using the native Object to JSON binding of [MOXy JAXB](http://www.eclipse.org/eclipselink/moxy.php).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://stackoverflow.com/questions/9111936/whats-the-purpose-of-minoccurs-nillable-and-restriction>

Nillable allows empty values. For example, if you have an Integer or a Date, if it's nillable, the XML tag could be empty. If it's not nillable but not required, the XML element would either have to exist with a valid content, or not exist at all; an empty tag wouldn't be valid

<https://jaxb.java.net/tutorial/>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://www.oracle.com/technetwork/articles/javase/index-140168.html>

Tutorial;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*