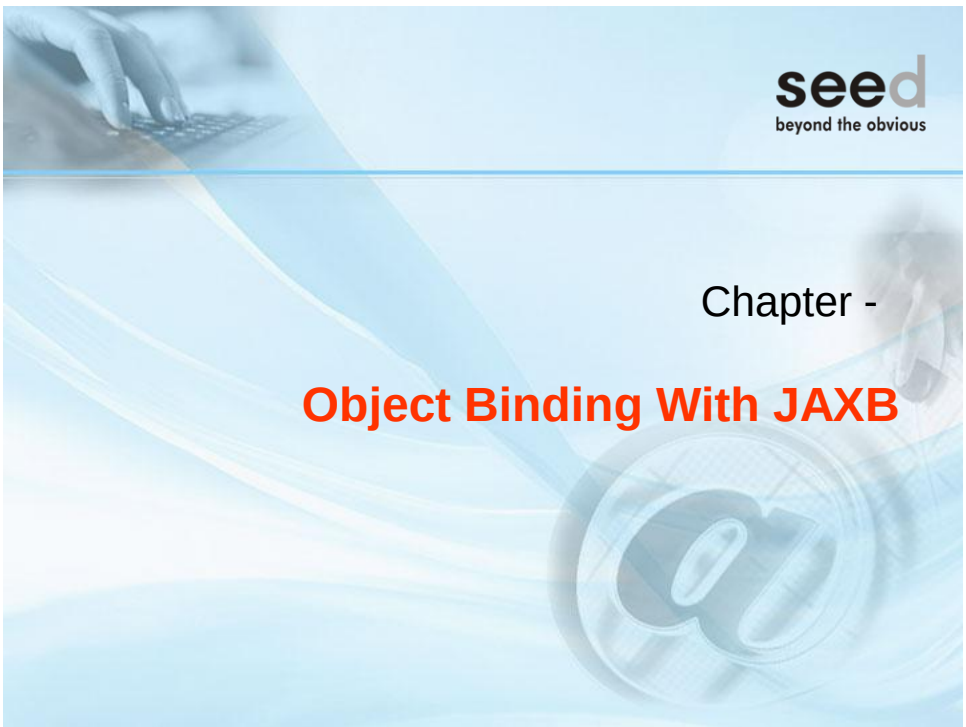
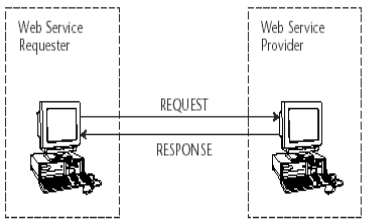


Objectives

- At the end of this chapter you will be able to understand
  - What JAXB is?
  - Why JAXB?
  - Architecture of JAXB 2.0

Introduction

- **X**ML is a simple, flexible, platform-independent language for representing structured textual information.
- The platform-independent nature of XML makes it an ideal vehicle for *exchanging* data across application components.
- When disparate application components exchange XML-based data, they do so because they want to process the exchanged data in some application-specific manner



seed
beyond the obvious

Introduction – Current Scenario

- This raises an interesting point:
  - although XML is ideal for exchanging data, processing XML content using the various APIs can be highly inefficient. Why is that so??

seed
beyond the obvious

Introduction – Current Scenario

- The answer is that most processing logic today resides within application components that are object oriented, whereas processing XML content is extremely procedural in nature.
  - Each component that wants to process some XML content has to not only be concerned that the content is *wellformed* but also that it conforms to some specific structure (or, in other words, is *valid* with respect to some schema).
  - Furthermore, once the component has verified that the XML content is well-formed and valid, it has to use an appropriate API to access the data embedded within the XML content.

Introduction – Current Scenario

- Of course, it can certainly do all that, but directly using these APIs within most object-oriented applications can be *highly inefficient from the point of view of encapsulation and code reuse*.

Introduction – JAXB

- To address the inefficiencies associated with directly processing XML content within object-oriented Java applications, we need a Java API that transparently maps XML content to Java objects and Java objects to XML content.
- *Java Architecture for XML Binding (JAXB)* is precisely such an API.

JAXB Overview

- The key to understanding JAXB is to focus on the following points:
  - Given an XML Schema document, an infinite number of XML documents can be constructed that would be valid with respect to the given schema.
  - Given a schema and an XML document that conforms to the given schema, an element within the given XML document must conform to a type component specified within the given schema.
  - What an object instance is to its corresponding class within Java, an element in an XML document is to an element declaration specified within the document's schema.

JAXB Overview continued…

- The key to understanding JAXB is to focus on the following points:
  - Each type component (with some exceptions) specified within a schema can be mapped to a Java class. This Java class may already exist as part of the Java platform, or it may need to be defined as a new class.
  - The process of binding schema type components to various Java class definitions is at the core of JAXB.

JAXB 2.0 - Architecture

- JAXB 1.0 was designed under a tight time constraint. As a result, the architects of this specification made a conscious decision to support the binding of only a subset of schema components to Java
- complete support was left to a later specification. JAXB 2.0 remedies the lack of complete schema support in JAXB 1.0 and adds binding support for missing schema components.

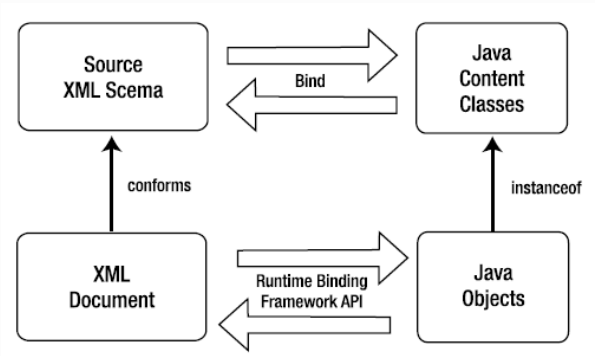
JAXB 2.0 - Architecture

- The following schema support was added to JAXB 2.0:
  - Element declarations using the ***substitutionGroup*** attribute, which is resolved to a predefined model group schema component (<xs:element @substitutionGroup>).
  - Wildcard schema components (xs:any and xs:anyAttribute).
  - Identity constraints used to specify uniqueness across multiple elements (xs:key, xs:keyref, and xs:unique).
  - Redefined XML Schema components using the redefine declaration (<xs:redefine>).
  - Notation XML Schema components (<xs:notation>).
  - The following schema attributes are supported:
    - complexType.abstract, element.abstract, element.substitutionGroup, xsi:type, complexType.block, complexType.final, element.block, element.final, schema.blockDefault, and schema.finalDefault.

JAXB 2.0 - Architecture

- The binding framework of JAXB 2.0 enhances the JAXB 1.0 unidirectional binding framework and adds support for bidirectional binding.
- JAXB 2.0 adds support for the binding of Java classes to XML Schema components, as shown

JAXB 2.0 - Architecture



seed
beyond the obvious

JAXB 2.0 - Annotations

- JAXB 2.0 relies on J2SE 5.0 annotations* to support bidirectional mapping between XML Schema and Java types.
- Annotations are used both in generated Java content classes and in Java classes as input to generate schema definitions.
- These binding annotations are defined in the ***javax.xml.bind.annotation*** package.

- * Annotations are a metadata facility for the Java programming language, defined as part of JSR-175 (http://www.jcp.org/aboutJava/communityprocess/review/jsr175/).

seed
beyond the obvious

JAXB 2.0 - Annotations

- Following Table lists some of the more commonly used annotations defined in the ***javax.xml.bind.annotation*** package…

seed
beyond the obvious

JAXB 2.0 - Annotations

Annotation Type	Description	Annotation Elements
`XmlAccessorType`	Specifies the default serialization of fields and properties	`AccessType.PUBLIC_MEMBER` maps only public fields and JavaBean properties. `AccessType.FIELDS` maps only fields. `AccessType.PROPERTIES` maps only JavaBeans properties. `AccessType.NONE` maps neither fields nor properties.
`XmlAttribute`	Maps a JavaBean property to an attribute	`name`: Attribute name. `namespace`: Attribute namespace. `required`: Specifies whether attribute is required; the default is false.
`XmlElement`	Maps a JavaBean property to an element	`defaultValue`: The default value of element. `name`: The element name. `namespace`: Target namespace of element. `nillable`: Specifies whether element is nillable; the default is `false`. `type`: Element type.
`XmlEnum`	Maps an enum to a simple type with enumeration	`value`: Enumeration value
`XmlList`	Maps a property to a list simple type	
`XmlRootElement`	Maps a class to root element	`name`: Local name of root element. `namespace`: Namespace of root element.
`XmlSchema`	Maps a package name to a XML namespace	`attributeFormDefault`: Specifies the value of the `attributeFormDefault` attribute. `elementFormDefault`: Specifies the value of the attribute `elementFormDefault`. `namespace`: XML namespace. `xmlns`: Maps namespace prefixes to namespace URIs.
`XmlType`	Maps a class to an XML Schema type, which may be a simple type or a complex type	`name`: Target namespace of the XML Schema type. `propOrder`: Specifies the order of XML schema elements when a class is mapped to a complex type.
`XmlValue`	Maps a class to an XML Schema complex type with `simpleContent` or an XML Schema simple type	

seed
beyond the obvious

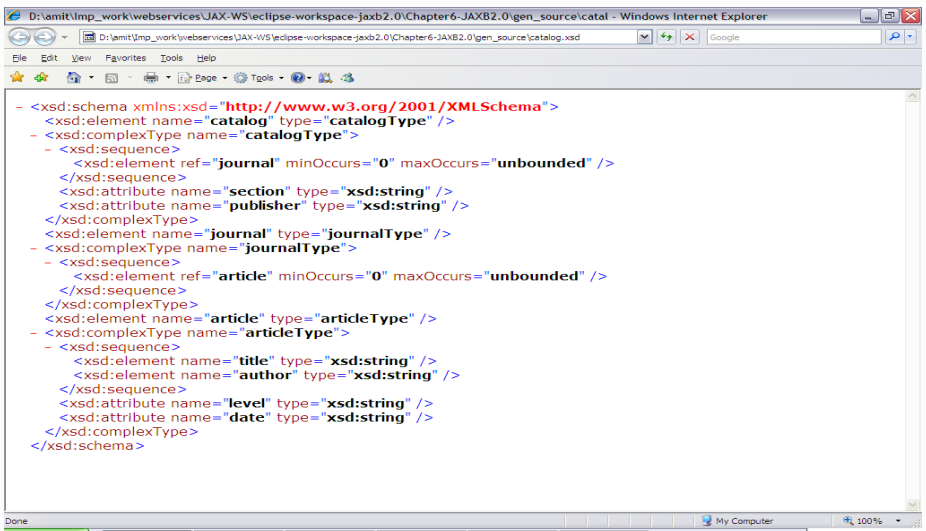
JAXB 2.0 - XML Schema Binding to Java Representation

- JAXB 2.0 specifies a default XML Schema to Java binding that can be overridden through external binding declarations.
- Conceptually, the JAXB 2.0 binding of XML Schema components to Java is similar to JAXB 1.0
- However, since JAXB 2.0 binding is based on J2SE 5.0, its Java representation uses Java 5 annotation tags and is much more compact than the JAXB 1.0 Java representation.

JAXB 2.0 – A Use Case

- Imagine a *website* selling various trade journals.
- This *website* offers a *web service* where associated
  - *publishers* can send *catalog* information about their *journals*.
  - The website provides an *XML Schema* that specifies the structure of an XML document containing catalog information.
  - This catalog schema defines a top-level ***catalog*** element. This catalog element can have *zero or more* ***journal*** elements, and each journal element can have *zero or more* ***article*** elements. Each of these elements defines relevant *attributes*. The elements are defined by reference to their associated types, which are defined separately.
- Lets say this schema file is ***catalog.xsd*** and is as shown…

JAXB 2.0 – A Use Case schema



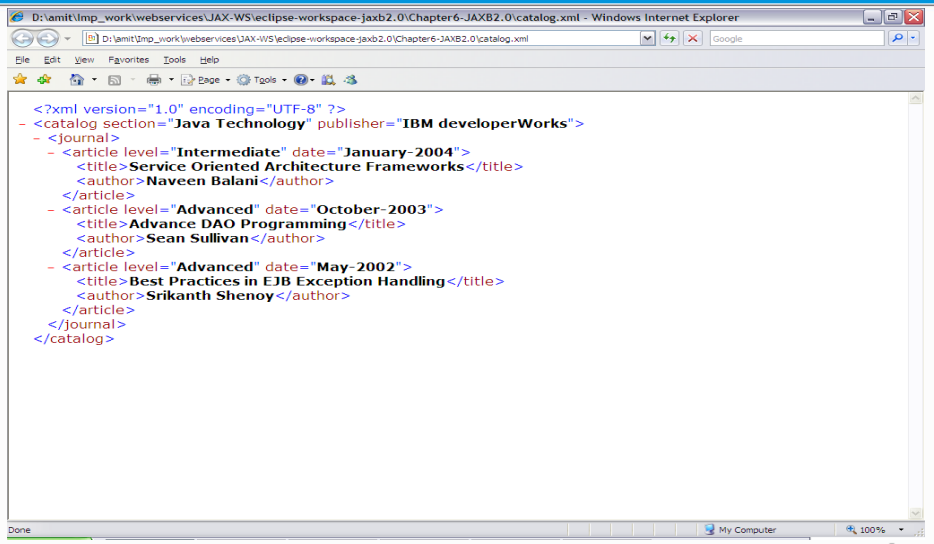
seed
beyond the obvious

JAXB 2.0 – A Use Case schema instance

- The web service client at the publisher must construct an XML document that conforms to the catalog schema shown and must send this document in a web service message.
- ***catalog.xml*** is one of the valid instance confirming the schema and is as shown…

seed
beyond the obvious

JAXB 2.0 – A Use Case schema instance



seed
beyond the obvious

JAXB 2.0 – A Use Case boundary

- The web service receiving this catalog information message needs to retrieve relevant element and attribute values from the message and store those values in a database.
- Here, we are not concerned with the aspects that deal with storing data in a database or that deal with the mechanics of assembling and transporting a web service message
- Our sole concern here is *marshaling* and *unmarshaling* the document and subsequently retrieving the relevant element and attribute values from the mapped Java objects.

seed
beyond the obvious

JAXB 2.0 – A Use Case objectives

- Our objectives are as follows:
  - Bind the catalog schema using the xjc compiler, and generate Java content classes representing the various schema components defined within the catalog schema.
  - *Marshal* and *unmarshal* the XML document
  - Retrieve the relevant element and attribute values from the mapped Java objects.

JAXB 2.0 – A Use Case Demo

- Demo Location:
  - eclipse-workspace-jaxb2.0\JAXB2.0

JAXB 2.0 – A Use Case Demo Observations

- The Java classes and interfaces are generated in the package ***generated***, by default.
- Fewer classes are generated with JAXB 2.0 than with JAXB 1.0
- For each ***xsd:complexType*** schema component, one value class gets generated, instead of an interface and an implementation class.
- For example, for the complex type ***catalogType***, in schema, the value class ***CatalogType.java*** gets generated as shown...

seed
beyond the obvious

JAXB 2.0 – A Use Case Demo Observations

```
<xsd:complexType name="catalogType">
  <xsd:sequence>
   <xsd:element ref="journal"  minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="section" type="xsd:string"/>
  <xsd:attribute name="publisher" type="xsd:string"/>
 </xsd:complexType>
```

```
package generated;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.AccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;
import generated.CatalogType;
import generated.JournalType;


@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "catalogType", propOrder = {
    "journal"
})
public class CatalogType {

    protected List<JournalType> journal;
    @XmlAttribute
    protected String publisher;
    @XmlAttribute
    protected String section;


    public List<JournalType> getJournal() {
        if (journal == null) {
            journal = new ArrayList<JournalType>();
        }
        return this.journal;
    }


    public String getPublisher() {
        return publisher;
    }
```

```
    public void setPublisher(String value) {
        this.publisher = value;
    }


    public String getSection() {
        return section;
    }
```

```
    public void setSection(String value) {
        this.section = value;
    }

}
```


seed
beyond the obvious

JAXB 2.0 – A Use Case Demo Observations

- Similarly, the value class ***JournalType.java*** gets generated for the complex type ***journalType***, and the value class ***ArticleType.java*** gets generated for the complex type ***articleType***.
- An ***ObjectFactory.java*** factory class gets generated, which consists of the create methods for each of the complex type and element declarations in the schema.
- For example, the ***ObjectFactory*** class method for the complex type ***catalogType*** is ***createCatalogType()***, and its return type is ***CatalogType***.
- The ***ObjectFactory*** class method for the element ***catalog*** is ***createCatalog(CatalogType)***, and its return type is ***JAXBElement<CatalogType>***

JAXB 2.0 – Marshaling an XML Document

- ***Marshaling*** a document means creating an XML document from a Java object tree.
- In the use case example, the *web services client* has to marshal the XML document.
- To marshal the example document, we need to follow these steps:

JAXB 2.0 – Marshaling an XML Document

- Create a *JAXBContext* object, and use this object to create a *Marshaller* object.
- Create an *ObjectFactory* object to create instances of the relevant generated *Java content classes*.
- Using the *ObjectFactory* object, create an object tree with *CatalogType* as the root object. Populate these tree objects with the relevant data using the appropriate setter methods.
- Create a *JAXBElement<CatalogType>* object from the *CatalogType* object.
- JAXBElement<CatalogType> represents the catalog element in XML document.

JAXB 2.0 – Unmarshaling an XML Document

- ***Unmarshaling*** means creating a Java object tree from an XML document.
- In the example use case, the website receives an XML document containing ***catalog*** information, and it needs to *unmarshal* this document before it can process the catalog information contained within the document.
- To unmarshal, we need to follow these steps:

JAXB 2.0 – Unmarshaling an XML Document

- The example XML document, ***catalog.xml*** , is the starting point for *unmarshaling*. Therefore, import catalog.xml to the Eclipse project.
- Create a ***JAXBContext*** object, and use this object to create an *UnMarshaller* object.
- The *Unmarshaller* class converts an XML document to a *JAXBElement* object of type ***CatalogType***.
- Create a ***CatalogType*** object from the *JAXBElement* object

seed
beyond the obvious

JAXB 2.0 – Binding Java Classes to XML Schema

- JAXB 2.0 supports bidirectional mapping between the XML Schema content and Java classes
- We can generate XML Schema content from a Java class using the JAXB 2.0 binding annotations. Therefore, we need to define an annotated class
- For example ***Catalog.java*** is a jaxb annotated class
- we can apply ***Catalog.java*** to the ***schemagen*** tool to generate a schema definition.

  Catalog.java

seed
beyond the obvious

JAXB 2.0 – Binding Java Classes to XML Schema

- In the Catalog.java class, import the ***javax.xml.bind.annotation*** package that includes the binding annotation types.
- Define the root element with the ***@XmlRootElement*** annotation.
- Create a complex type using the @***XmlType*** annotation:

```
@XmlRootElement
@XmlType(name="", propOrder={"publisher", "edition", "title",
    "author"})
```

- We specify the annotation element ***name*** as an empty string because the complex type is defined anonymously within an element.
- We specify the element order using the ***propOrder*** annotation element.

seed
beyond the obvious

JAXB 2.0 – Binding Java Classes to XML Schema

- In the ***Catalog*** class, define constructors for the class, and define the different JavaBean properties (publisher, edition, title, author).
- The root element ***catalog*** has an attribute ***journal***.

Define the ***journal*** attribute using the ***@XmlAttribute annotation:***

```
@XmlAttribute
public String journal;
```

- We also need to define getter and setter methods for the different properties and the journal attribute.

seed
beyond the obvious

Quick Recap . . .

- JAXB 2.0 provides following advantages :
  - Support for all the schema constructs
  - A relatively compact binding of a schema definition to Java content classes
  - Bidirectional mapping between schema definition and Java types
- It is strongly recommend using JAXB 2.0, unless we explicitly need to stay with JAXB 1.0, such as for backward compatibility.