



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

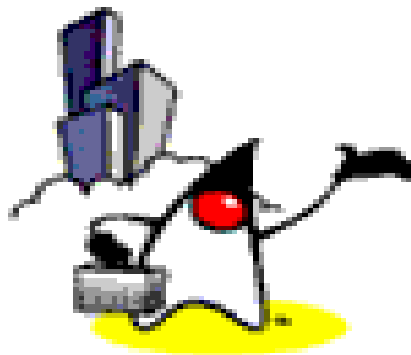


Agenda

- XML Refresher
- What is and Why JAXB?
- How to use JAXB?
- JAXB architecture
- Binding process
- Custom binding
- Runtime operations
- JAXB Roadmap



XML Refresher



Key Aspects of XML

- XML is a platform independent means of **structuring information**
- An XML document is a tree of elements
- An Element
 - Can have attributes (key-value)
 - Can contain other elements, text or a mixture of both
- An Element refers to another Element via identifier/key attributes (arbitrary graph structures)

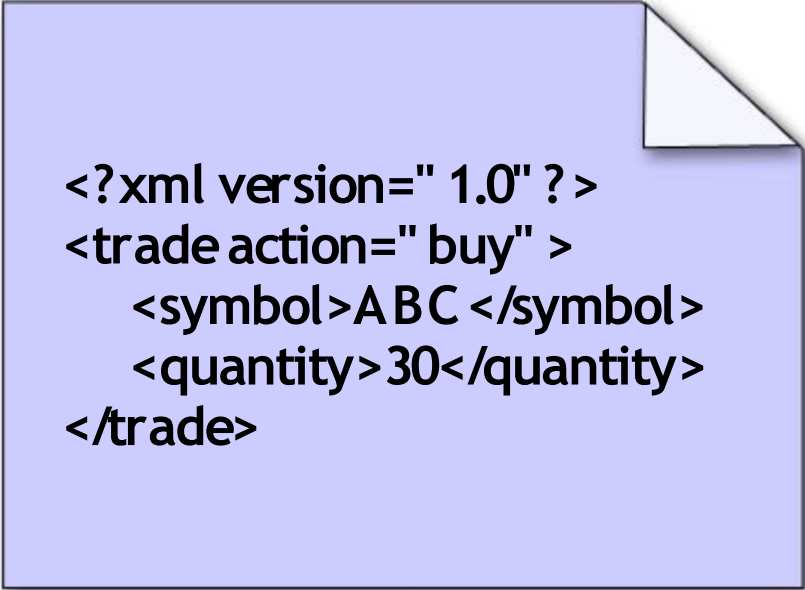
XML Documents: Well-formed & Valid

- Need only be **well formed**
 - Not necessarily Valid
- For meaningful exchange, however:
 - Structure and Content must be
 - Described
 - Constrained
 - Accomplished by use of a schema
 - A schema specifies **syntax**
 - A scheme **validates** a XML document

Schema types

- Content-oriented
 - Closest to programming data structures
 - Element, type defs, attribute names semantically reflect content they represent
- Structure-oriented
 - Store content in form convenient for presentation,
 - Mixes content and presentation
 - Examples: HTML, an invoice, a book
- Presentation-oriented

Sample Document Instance



```
<?xml version=" 1.0" ? >  
<trade action=" buy" >  
  <symbol>ABC </symbol>  
  <quantity>30</quantity>  
</trade>
```

Languages for XML Schemas

- XML 1.0 Document Type Definitions (DTDs)
 - Fairly weak constraints
 - No data types or complex structural relationships
- W3C XML Schema Definition Language Recommendation (2, May, 01)
- Many Others
 - RELAXNG, SOX, XDR, TREX

W3C XML Schema Definition for Sample Document Instance

```
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema" >
  <xsd:complexType name="tradeType">
    <xsd:sequence>
      <xsd:element name = "symbol" type = "xsd:string"/>
      <xsd:element name = "quantity" type = "xsd:int"/>
    </xsd:sequence>
    <xsd:attribute name = "action" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:NMTOKEN">
          <xsd:enumeration value = "buy"/>
          <xsd:enumeration value = "sell"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:element name = "trade" type="tradeType"/>
</xsd:schema>
```

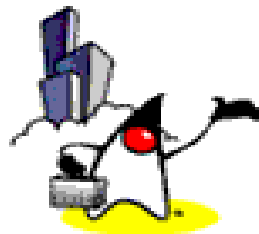
```
<!--document instance-->
<?xml version=" 1.0" ?>
<trade action=" buy" >
  <symbol>ABC </symbol>
  <quantity>30</quantity>
</trade>
```

XML Processing Options

- SAX
 - Stateless event driven
- DOM
 - Memory intensive parse tree
- StAX
 - Streaming API for XML
- JAXB
 - Static binding of XML schema to JavaBeanTM components



What is & Why JAXB?



What Is JAXB?

- Provides **API, tools, and a framework** that automate the mapping between XML documents and Java objects
 - Provides compiler that compiles XML schema to Java classes
- What should the classes be?
 - Obvious cases: int, String, Date, List, ...
 - Classes can be generated **from the schema** for the other cases
 - A Java™ technology-level binding of the schema

Things You can do during Runtime

- **Unmarshal** XML content (XML document instance) to Java representations
- **Access, update and validate** the Java representation against schema constraints
- **Marshal** the Java representation of the XML content into XML content

Why JAXB?

- Provides an efficient and **standard** way of mapping between XML and Java code
 - Programmers don't have to create application specific Java objects anymore themselves
 - Programmers do not have to deal with XML structure, instead deal with meaning business data
 - *getPerson()* method as opposed to *getAttributes()*
- In a sense JAXB is high-level language while JAXP/SAX/DOM are assembly language for XML document management

Why JAXB?

- Issues with DOM / SAX model
 - Why do I have to walk a **Parse Tree** containing much more than just application data?
 - `getNodeName()`, `getNodeValue()`, `getNodeType()`
 - Why do I have to write **EventHandlers** to map XML content to JavaTM classes?
- Value proposition of JAXB
 - JAXB **automates** XML to JavaTM binding so you can easily access your data

JAXB versus DOM

- Both JAXB and DOM create in-memory content tree
- In JAXB,
 - Content tree is specific to a **specific source schema**
 - Does not contain extra tree-manipulation functionality
 - Allows access to its **data** with the derived classes' accessor methods
 - “**Data-driven**” as opposed to “XML document driven”
 - Content tree is not created dynamically, thus uses memory efficiently

JAXB Design Goals

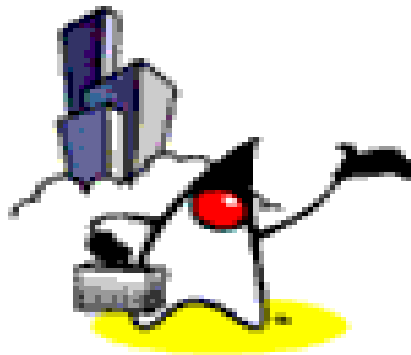
- Easy to use
 - Lower “barrier to entry” to manipulating XML documents within Java programs
 - Don't have to deal with complexities of SAX and DOM
- Customizable
 - Sophisticated applications sometimes require fine control over the structure and content of schema-derived classes
 - Allows keeping pace with schema evolution

JAXB Design Goals

- Portable
 - JAXB components can be replaced without having to make significant changes to the rest of the source code
- Validation on demand
 - Validate the tree against the constraints in the source schema
- Clean “round-tripping”
 - Converting a Java content tree to XML content and back to Java content again should result in equivalent Java content trees before and after the conversion



How to Use JAXB?



Steps of Building & Using JAXB Applications (Compile time)

1. Develop or obtain XML schema

- Optionally annotate the schema with binding customizations if necessary (or place them in an external binding file)

2. Generate the Java source files

- By compiling the XML Schema through the binding compiler

3. Develop JAXB client application

- Using the Java content classes generated by the binding compiler along with the [javax.xml.bind](#) JAXB interfaces

4. Compile the Java source codes

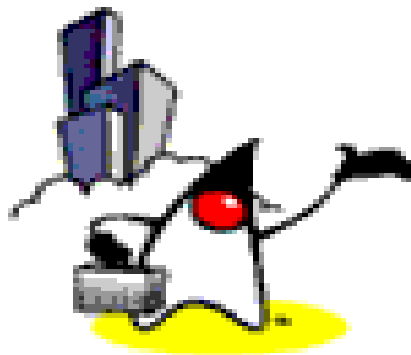
Steps of Building & Using JAXB Applications (Runtime)

5. With the classes and the binding framework, write Java applications that:

- Build object trees representing XML data that is valid against the XML Schema by
 - either **unmarshalling the data from an XML document**
 - or **instantiating** the classes you created
- Access and modify the data
- Optionally validate the modifications to the data relative to the constraints expressed in the XML Schema
- Marshal in-memory data to a new XML document



Schema Languages

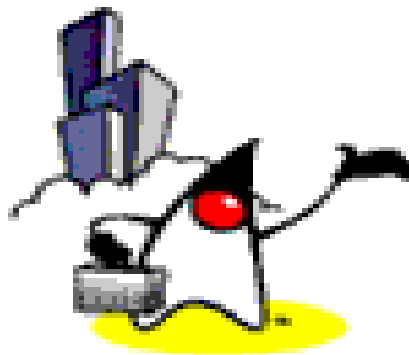


Schema Languages that JAXB Supports

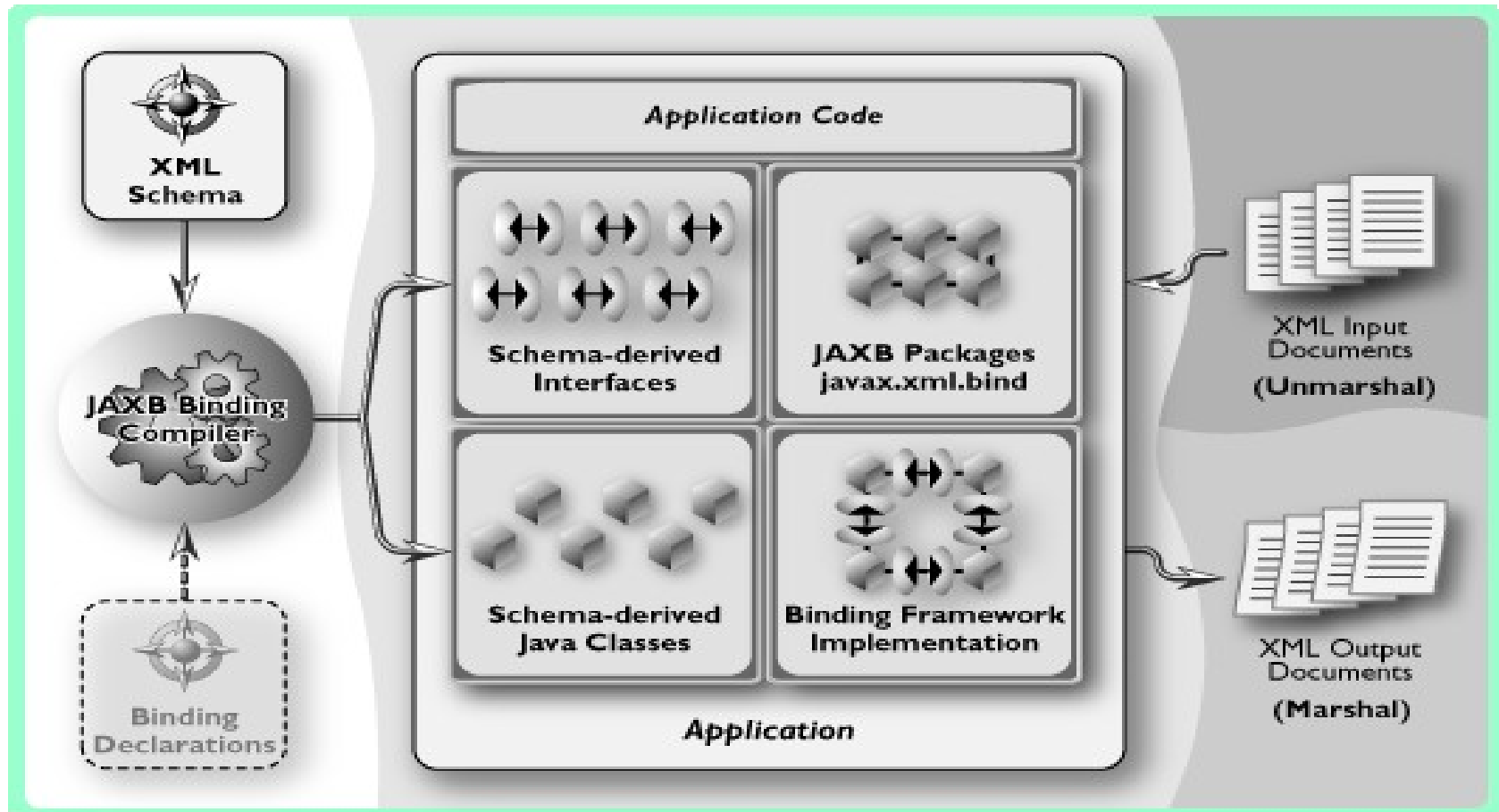
- For JAXB 1.0 specification, only **W3C XML Schema** is supported
 - No DTD support
- Vendor extension of JAXB 1.0 might support other schema languages
 - JAXB RI 1.0.1 (compliant with JAXB spec 1.0) from Java WSDP supports DTD and RELAX NG



JAXB Architecture



JAXB Architecture

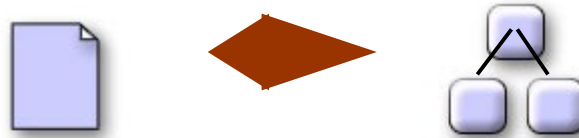


XML Data Binding Facility

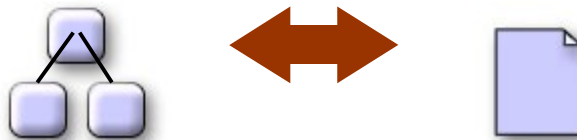
- Binding compiler
 - Binds schema components to derived classes
- Schema derived interfaces and classes
 - A **content tree** is a tree of in-memory instances
 - The **methods** provide access to the content of the corresponding schema component
- Binding Runtime API
 - Provide runtime XML-enabling operations (unmarshalling, marshalling, validating) on content trees through schema-derived classes
 - Implementation of [javax.xml.bind](#)

Binding Runtime API

- The basic binding operations provided are:
 - **Unmarshal** XML document to a content tree

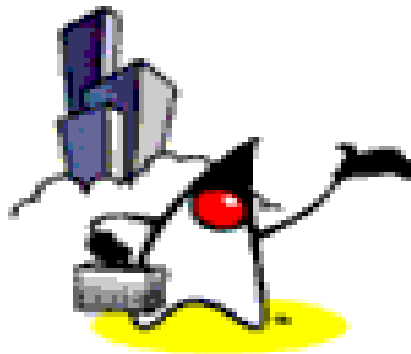


- **Validate** a content tree based on constraints expressed within the original schema
- **Marshal** a content tree to an XML document





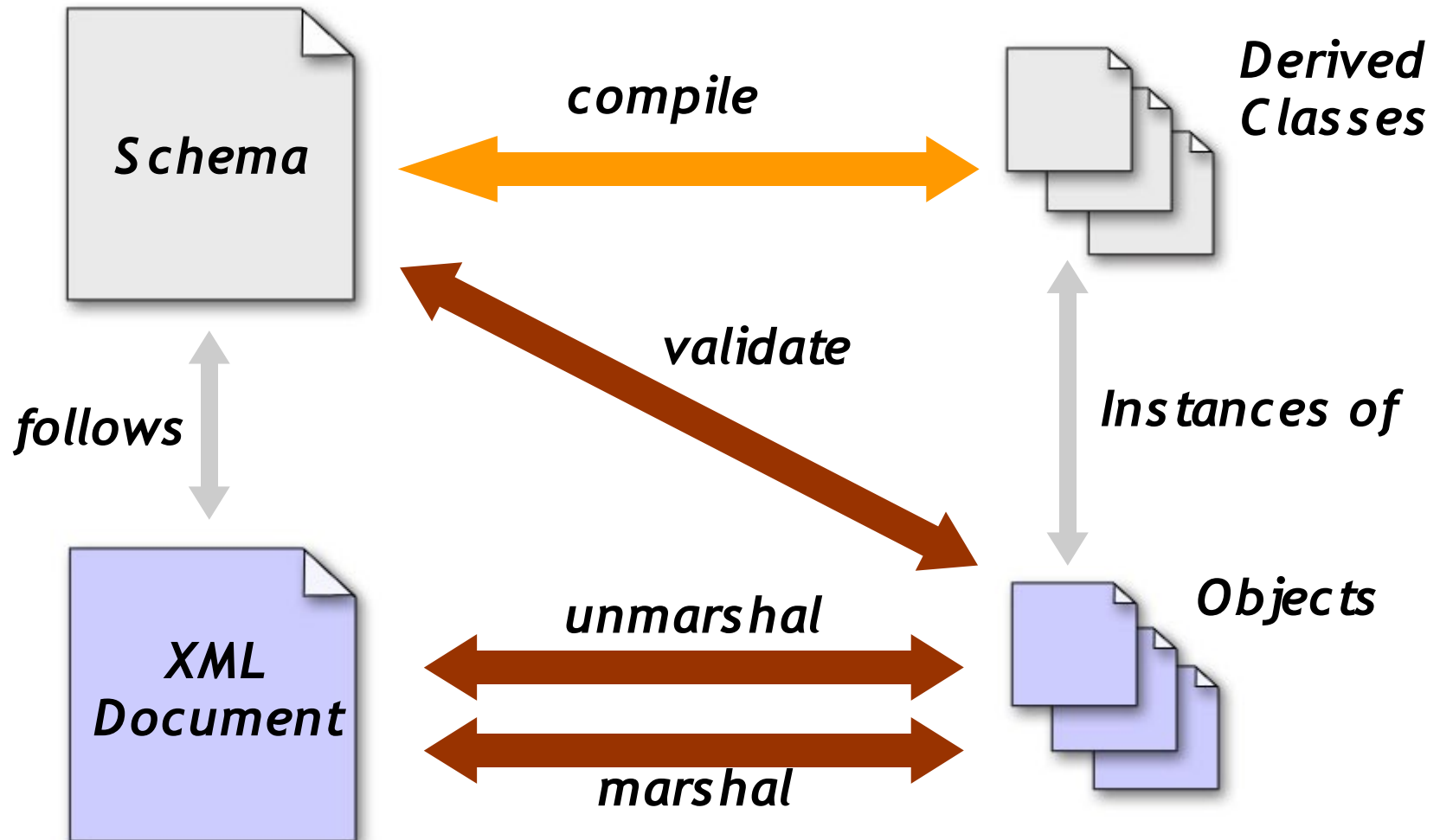
Binding Process

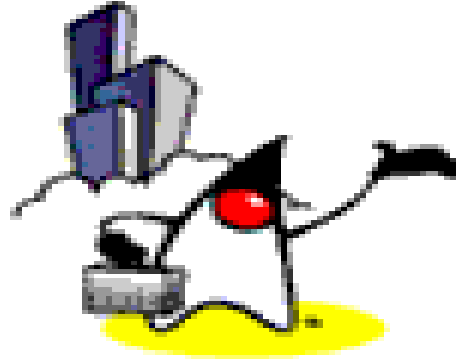


Binding Process: 2 Phases

- Design time
 - Generate JAXB classes from a source schema
 - Compile the JAXB classes
 - Build an application that uses these classes
- Run time (Binding runtime)
 - Unmarshal XML content tree
 - Process (Access & Modify) XML content tree
 - Validate XML content tree
 - Marshal XML content tree

The Binding Life Cycle





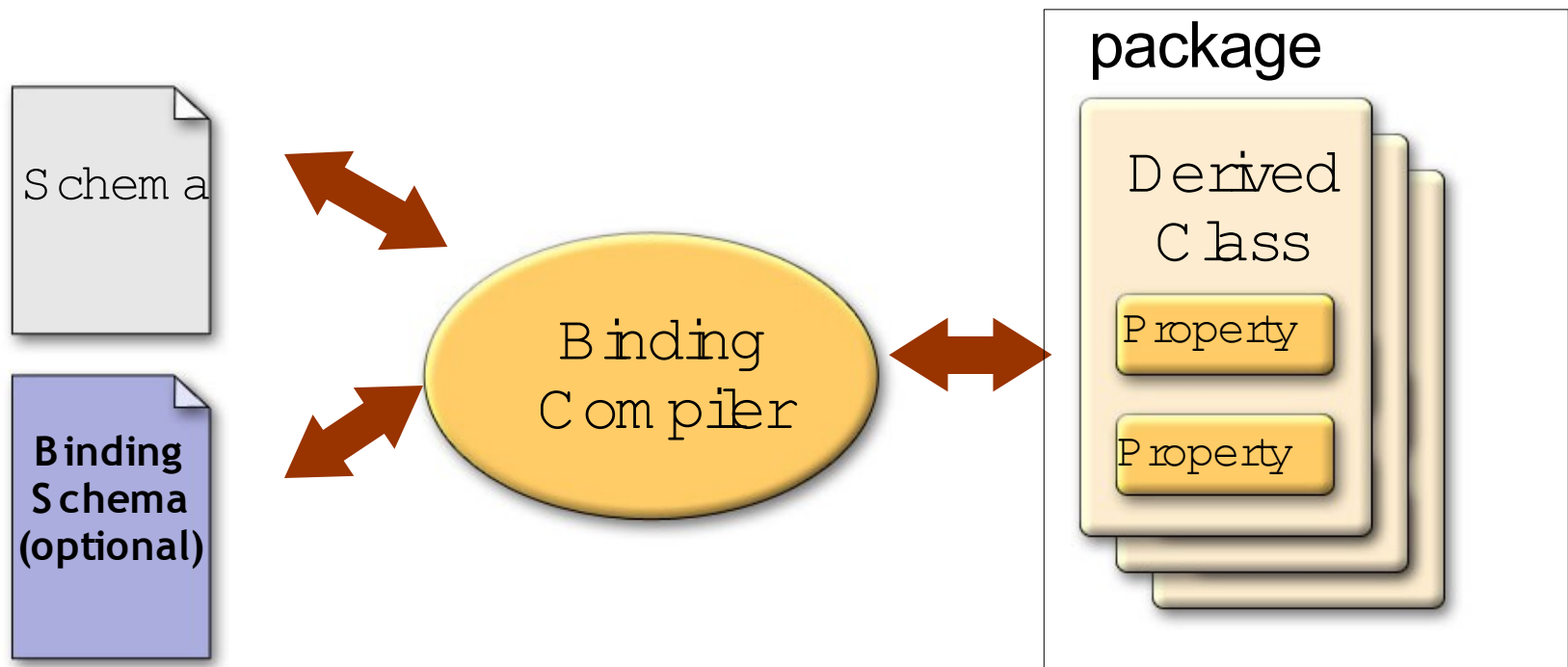
Binding Process

Design Time Binding

Process

Role of Binding Compiler

- Bind **target XML namespace** to **package**
- Bind **element or complex type** to **derived class**
- Bind **attribute or leaf element** to a **property**

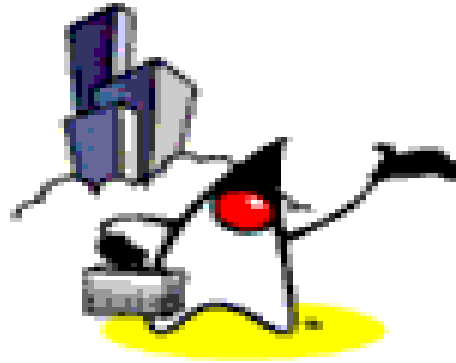


Binding Rules

- Default rules
 - Specified in the JAXB spec
 - Binding compiler uses **default rules** for components of source schema not mentioned explicitly in custom binding declaration
- Custom rules (**Custom Binding Declaration**)
 - Can be “inline”ed in XML schema document or
 - Separate custom binding declaration file

Derived Classes Should:

- Observe standard Java API design guidelines and naming conventions
 - Algorithm specified to map legal XML names to legal Java identifiers
- Conceptually match schema
 - Structurally
 - Preserve validation constraints
- Hide the plumbing as much as possible
 - Minimize required usage of SAX or DOM to accessing dynamic content



Binding Process

Default Binding

Default XML to Java Binding

- Simple Type Definitions
- Default Data Type Bindings
- Default Binding Rules Summary

Simple Type Definitions

- Typically binds to Java Property
- Java Property attributes
 - Base type
 - Collection type
 - Predicate

Default Data Type Binding

XML Schema Type

- xsd:string
- xsd:integer
- xsd:int
- xsd:long
- xsd:short
- xsd:decimal
- xsd:float
- xsd:double
- xsd:boolean
- xsd:byte
- xsd:QName

Java Data Type

- java.lang.String
- java.math.BigInteger
- int
- long
- short
- java.math.BigDecimal
- float
- double
- boolean
- byte
- javax.xml.namespace.QName

me

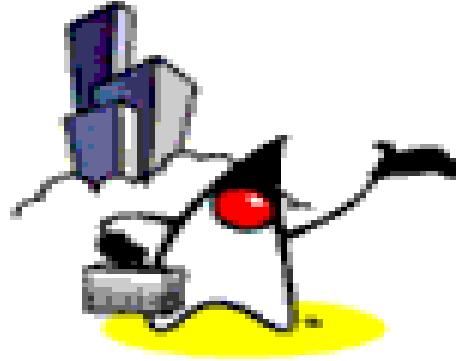
Default Data Type Binding

XML Schema Type

- xsd:dateTime
- xsd:base64Binary
- xsd:hexBinary
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte
- xsd:time
- xsd:date
- xsd:anySimpleType

Java Data Type

- java.util.Calendar
- byte[]
- byte[]
- long
- int
- short
- java.util.Date
- java.util.Date
- java.lang.String



Binding Process

Default Binding Rules & Examples

Default Binding Rules: Global Element

From

- A global element declaration to a Element interface
- Local element declaration that can be inserted into a general content list
- Attribute

To

- Java Element interface
- Java Element interface
- Java property

Default Binding Example: Global Element

XML Schema

- `<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">`
- `<xsd:element name="purchaseOrder" type="PurchaseOrderType" />`
- `<xsd:element name="comment" type="xsd:string"/>`

JAXB Binding

- (nothing)
- `PurchaseOrder.java`
- `Comment.java`

PurchaseOrder.java

```
package primer.po;
```

```
/**
```

```
 * Java content class for purchaseOrder element declaration.
```

```
 * <p>The following schema fragment specifies the expected  
  content
```

```
 * contained within this java content object.
```

```
 * <p>
```

```
 * <pre>
```

```
 * &lt;element name="purchaseOrder"  
  type="{PurchaseOrderType"/>
```

```
 * </pre>
```

```
 *
```

```
 */
```

```
public interface PurchaseOrder
```

```
    extends javax.xml.bind.Element, primer.po.PurchaseOrderType {  
    }
```

Default Binding Rules: Named ComplexType

From

- XML Namespace URI
- Named complexType
- Anonymous inlined type definition of an element declaration
- A named simple type definition with a basetype that derives from "xsd:NCName" and has enumeration facet

To

- Java package
- Java content interface
- Java content interface
- typesafe enum class

Default Binding Example: Named ComplexType

XML Schema

```
<xsd:complexType  
  name="PurchaseOrderType">  
  <xsd:sequence>  
    <xsd:element name="shipTo"  
      type="USAddress"/>  
    <xsd:element name="billTo"  
      type="USAddress"/>  
    <xsd:element ref="comment"  
      minOccurs="0"/>  
    <xsd:element name="items"  
      type="Items"/>  
  </xsd:sequence>  
  <xsd:attribute name="orderDate"  
    type="xsd:date"/>  
</xsd:complexType>
```

JAXB Binding

PurchaseOrderType.java

PurchaseOrderType.java

```
package primer.po;
public interface PurchaseOrderType {

    primer.po.Items getItems();
    void setItems(primer.po.Items value);

    java.util.Calendar getOrderDate();
    void setOrderDate(java.util.Calendar value);

    java.lang.String getComment();
    void setComment(java.lang.String value);

    primer.po.USAddress getBillTo();
    void setBillTo(primer.po.USAddress value);

    primer.po.USAddress getShipTo();
    void setShipTo(primer.po.USAddress value);
}
```

Default Binding Example: ComplexType

XML Schema

```
<xsd:complexType name="USAddress" > USAddress.java
  <xsd:sequence>
    <xsd:element name="name"
      type="xsd:string"/>
    <xsd:element name="street"
      type="xsd:string"/>
    <xsd:element name="city"
      type="xsd:string"/>
    <xsd:element name="state"
      type="xsd:string"/>
    <xsd:element name="zip"
      type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country"
    type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

JAXB Binding

USAddress.java

USAddress.java

```
package primer.po;  
public interface USAddress {  
  
    java.lang.String getState();  
    void setState(java.lang.String value);  
  
    java.math.BigDecimal getZip();  
    void setZip(java.math.BigDecimal value);  
  
    java.lang.String getCountry();  
    void setCountry(java.lang.String value);  
  
    java.lang.String getCity();  
    void setCity(java.lang.String value);  
  
    java.lang.String getStreet();  
    void setStreet(java.lang.String value);  
  
    java.lang.String getName();  
    void setName(java.lang.String value);  
}
```


Derived Interfaces

- Each complex type definition (Named ComplexType) mapped to a derived interface and implementation class
 - Enable user implementation of derived interface via binding schema
- Type definition derivation hierarchy mapped to Java inheritance class hierarchy
 - Natural mapping of "derivation by extension"
 - No plans to enforce "derivation by restriction"

Default Binding Rules: Attribute

From	To
<ul style="list-style-type: none">• A global element declaration to a Element interface• Local element declaration that can be inserted into a general content list• <u>Attribute</u>	<ul style="list-style-type: none">• <u>Java Element interface</u>• <u>Java Element interface</u>• <u>Java property</u>

Derived Classes: Properties

- Fine-grained XML components bound to a property
 - Accessed by setter and getter methods similar to JavaBeans™ property accessors
 - Optional validation checking by setter
- Three Core Property types
 - Simple Bean design pattern
 - Indexed Bean design pattern
 - Collection Map to [java.util.List](#) types

Example: Default XML Binding

XML Schema

```
<xsd:complexType
name="trade">
  <xsd:sequence>
    <xsd:element name="symbol"
                  type
="xsd:string"/>
  </sequence>
    <xsd:attribute name="quantity"
                  type ="xsd:int"/>
  </xsd:complexType>
```

Derived Class

```
public interface
Trade {

  String getSymbol();
  void
setSymbol(String);

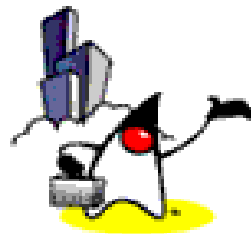
  int   getQuantity();
  void setQuantity(int);
}
```

Property Basics

- Invoking `setX(null)` discard property's set value
- getter method returns
 - property's set value, if it is set with non-null
 - schema specified default value, if existed
 - Java default initial value for property's base type
- Additional Property methods
 - `isSetX()` returns true if property's value is set
 - Not generated by default, not all apps need this



Custom Binding



Why Custom Binding?

- Customize your generated JAXB classes beyond the XML-specific constraints in an XML schema
 - Java-specific refinements such as class and package name mapping if default name-mapping is not adequate
- Resolve name collisions
 - Caused by XML Schema symbol spaces and foreign element/attributes references
 - User provides semantically meaningful and deterministic name resolution

Things you want customize

- Configuration level
 - Override built-in datatype bindings
- Property
 - Collection as indexed or [java.util.List](#)
 - Manipulate property's default value
- Class
 - Bind nested model group to a JavaBean component
 - Complete JavaBean™ component binding
 - Event notification model

Customization Mechanism

- One standard customization language
xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
- Customize a schema using either
 - **Inline** annotation
 - **External** binding declaration
- Extensible
 - Uses same technique as XSLT 1.0 to introduce vendor extension customization namespace(s)

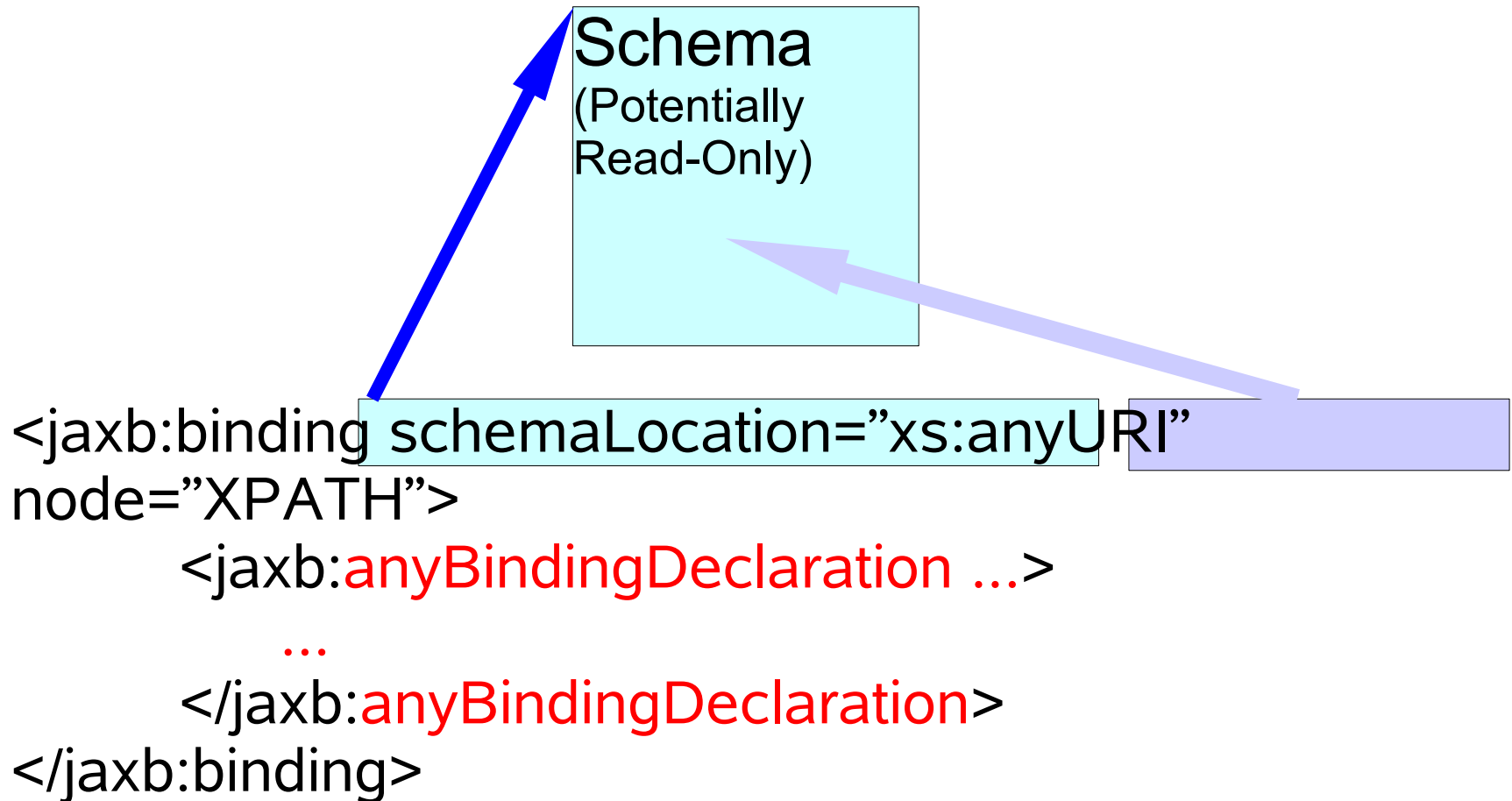
Name Collision Resolution

```
<xs:schema xmlns="X" targetNamespace="X"
            xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
            jxb:version="1" >
  <xs:annotation><xs:appinfo>
    <jxb:schemaBindings>
      <jxb:nameXmlTransform>
        <jxb:elementName suffix="Element"/>
      </jxb:nameXmlTransform>
    </jxb:schemaBindings>
  </xs:appinfo></xs:annotation>
  <xs:complexType name="trade"/>
  <xs:element name="trade" type="trade"/>
</xs:schema>
```

In Package X

```
import javax.xml.bind.Element;
public interface Trade {};
public interface TradeElement implements Element {}
```

External Binding Declaration



Scope of Custom Binding

- When a customization value is defined in a binding declaration, it is associated with a scope
- 4 scopes
 - Global: A customization value defined in `<globalBindings>`
 - Schema: A customization value defined in `<schemaBindings>`
 - Definition: A customization value in binding declarations of a type definition and global declaration
 - Component: A customization value applied only to the schema element

Customization Scopes

`<jaxb:globalBindings>` - Applies to All XML namespaces

`<jaxb:schemaBindings>` - One per XML namespace

Definition scope – Per component define

Component Scope (Particle and refs)

Binding declaration(s)

Best Practice Guidelines

- In most cases, default binding should be sufficient
- Use default binding rules whenever possible
 - Better maintainability when Schema changes

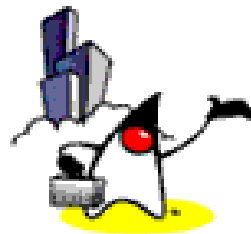
Vendor Extensions

- Why?
 - Satisfy app requirements not met by spec yet
 - Allows for experimentation
 - Enabled by `<jaxb:extensionBindingPrefixes>`
- JAXB v1.0.1 RI Implementation
 - `<xjc:serializable>`
 - Generated classes can be passed via RMI
 - `<xjc:superClass>`
 - Application-level base class for schema-derived classes
 - `<xjc:dom>`
 - Bind schema component to a DOM tree



JAXB

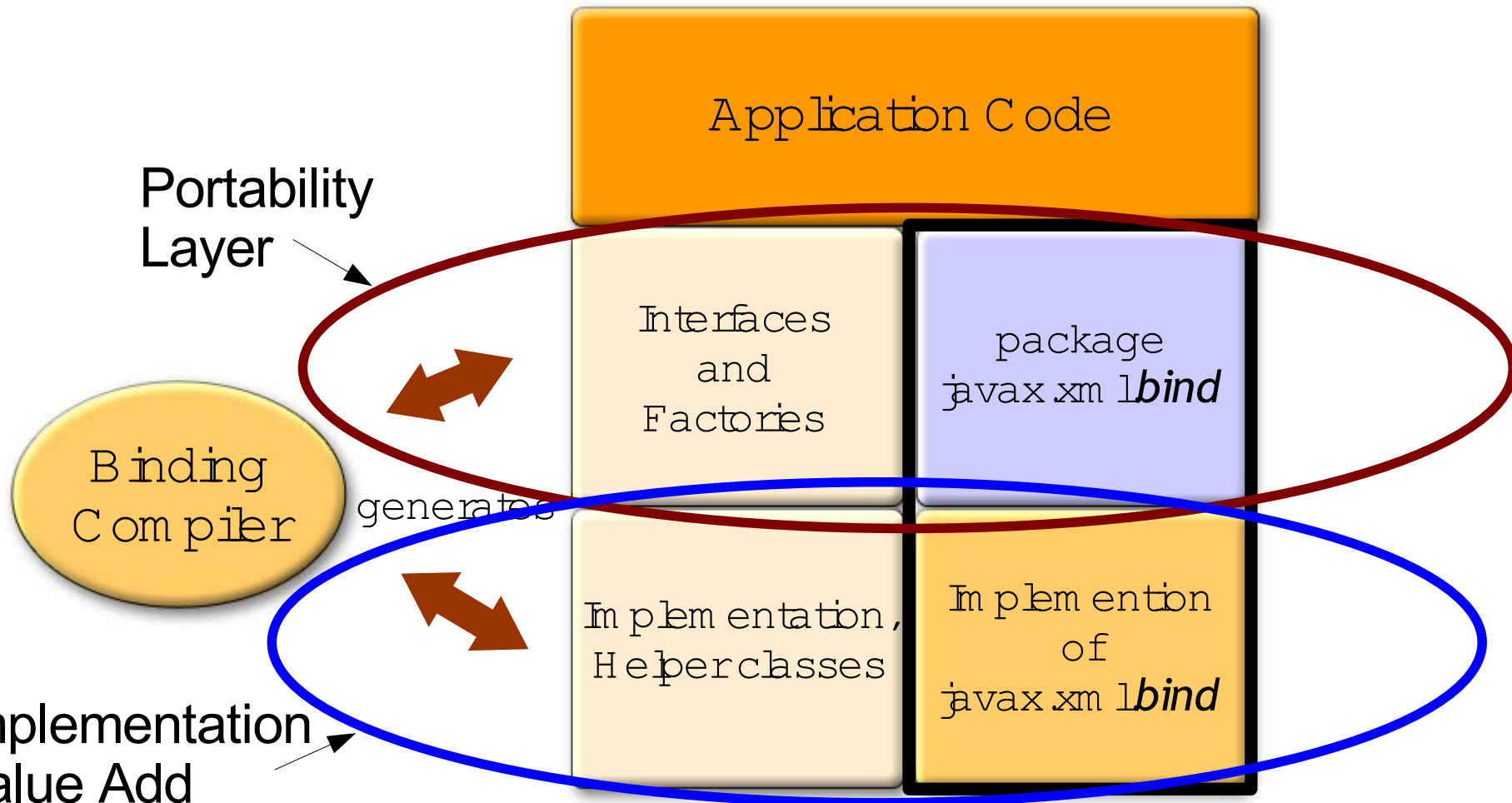
Runtime Operations

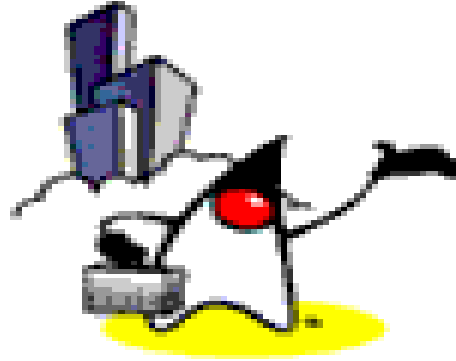


JAXB Runtime Operations

- Provide the following functionality for schema-derived classes
 - Unmarshal
 - Process (access or modify)
 - Marshal
 - Validation
- A factory generates **Unmarshaller**, **Marshaller** and **Validator** instances for JAXB technology-based applications
 - Pass content tree as parameter to Marshaller and Validator instances

Runtime Framework

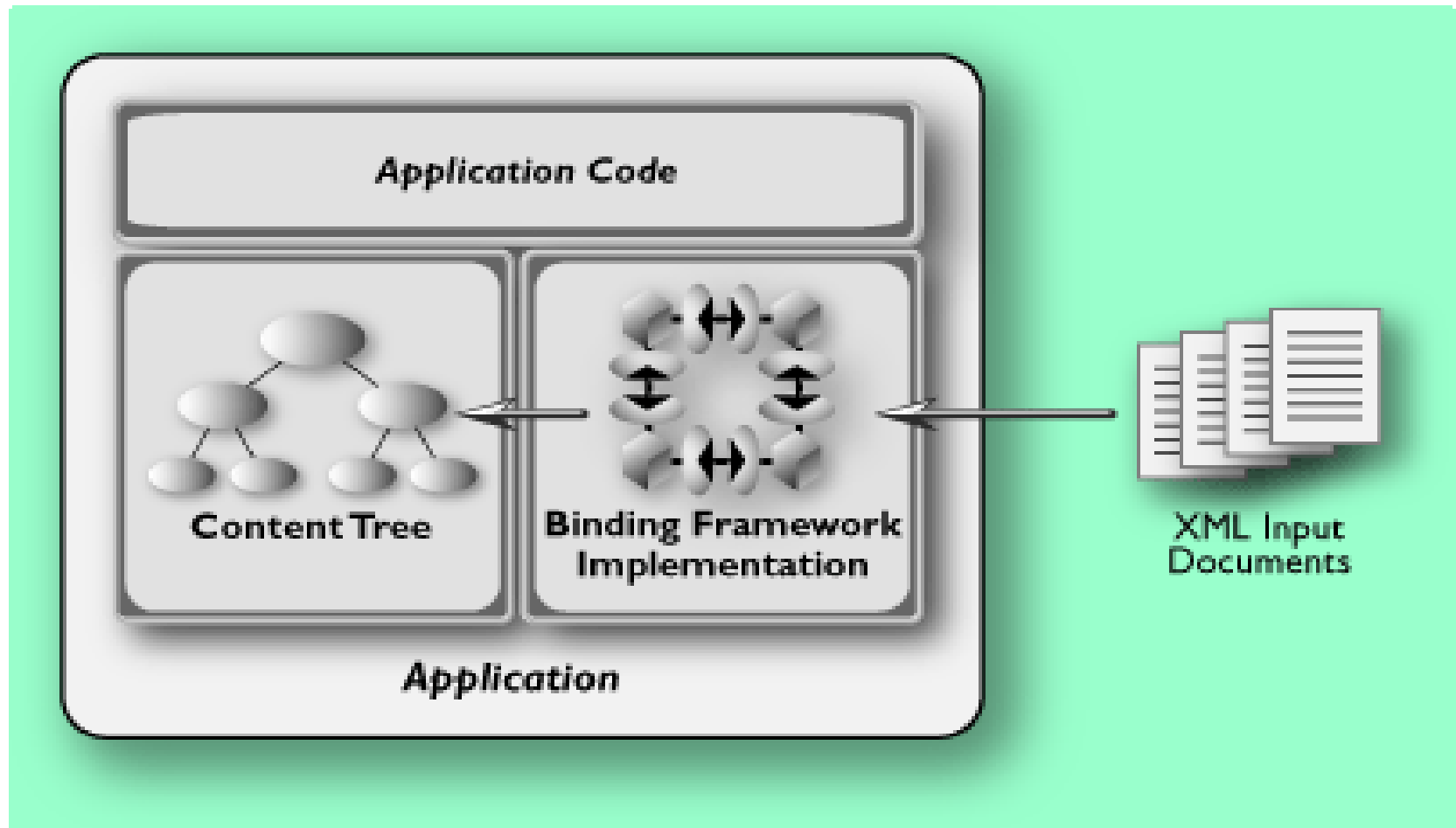




JAXB Runtime Operations

Unmarshalling

UnMarshalling Architecture



UnMarshalling

- **Generates content tree** from XML document instance through JAXB binding framework
- Sources for unMarshalling can be
 - Files/documents
 - InputStream
 - String buffers
 - DOM nodes
 - SAX Sources

javax.xml.bind.JAXBContext

- Provides **entry point** to the JAXB API
 - Provides an abstraction for managing the XML/Java binding information necessary to implement the unmarshal, marshal and validate operations
- Created via **newInstance(contextPath)**
 - **contextPath** contains a list of Java package names that contain schema derived interfaces and classes

```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo:com.acme.bar" );
```

- Unmarshaller, Marshaller, Validator object are created from **JAXBContext** object

javax.xml.bind.Unmarshaller

- Java Interface
- Governs the process of deserializing XML data (XML document instance) into newly created Java content tree
- Optionally validates XML data as it is unmarshalled

Unmarshalling from a File

```
// Create JAXBContext object
```

```
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );
```

```
// Create Unmarshaller object
```

```
Unmarshaller u = jc.createUnmarshaller();
```

```
// Unmarshall a XML document which is in the form of File
```

```
Object o = u.unmarshal( new File( "example.xml" ) );
```


Unmarshalling from an InputStream

```
InputStream is = new FileInputStream( "example.xml" );  
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();  
Object o = u.unmarshal( is );
```

Unmarshalling from a URL

```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );
```

```
Unmarshaller u = jc.createUnmarshaller();
```

```
URL url = new URL( "http://beaker.east/example.xml" );
```

```
Object o = u.unmarshal( url );
```

Unmarshalling from a StringBuffer

```
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();  
StringBuffer xmlStr = new StringBuffer( "<?xml  
    version='1.0'?>..." );  
Object o = u.unmarshal( new StreamSource( new StringReader(  
    xmlStr.toString() ) ) );
```

Unmarshalling from a `org.w3c.dom.Node`

```
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();
```

```
DocumentBuilderFactory dbf =  
    DocumentBuilderFactory.newInstance();  
dbf.setNamespaceAware(true);  
DocumentBuilder db = dbf.newDocumentBuilder();  
Document doc = db.parse(new File( "example.xml" ));
```

```
Object o = u.unmarshal( doc );
```

Unmarshalling from a `javax.xml.transform.sax.SAXSource`

```
XMLReader xmlReader = saxParser.getXMLReader();  
SAXSource source =  
    new SAXSource( xmlReader, new InputSource( "http://..." ) )  
    ;
```

```
// Setup JAXB to unmarshal  
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();  
ValidationEventCollector vec = new ValidationEventCollector();  
u.setEventHandler( vec );
```

```
// turn off the JAXB provider's default validation mechanism to  
// avoid duplicate validation  
u.setValidating( false )
```

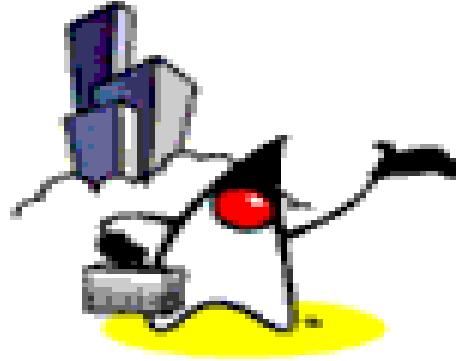
```
// unmarshal  
Object o = u.unmarshal( source );
```

Validation During Unmarshalling

- Flexibility to enable/disable validation
 - It is a waste of time to validate valid documents
- No longer required to terminate upon encountering first validation failure
 - Implementation decision when to terminate
 - Enable fixing minor validation constraint errors
- Increased flexibility comes with a cost, less deterministic behavior

Example: Turn off Validation During Unmarshalling

```
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();  
u.setValidating(false);  
Object o = u.unmarshal( new File( "example.xml" ) );
```



JAXB Runtime Operations

Creating In-memory

Content

Programmatically

Creating Content Tree via Programmatic Factory

- Another way of creating a content tree
 - You don't need XML document
 - Application needs to have access and knowledge about each of the schema derived ObjectFactory classes that exist in each of java packages contained in the contextPath
 - For each schema derived java class, there will be a static factory method that produces objects of that type
 - Once the client application has an instance of the the schema derived object, it can use the mutator methods to set content on it

Example Code: Programmatic generation of Content Tree

```
// Assume that after compiling a schema, you have a package  
// com.acme.foo that contains a schema derived interface  
// named PurchaseOrder.
```

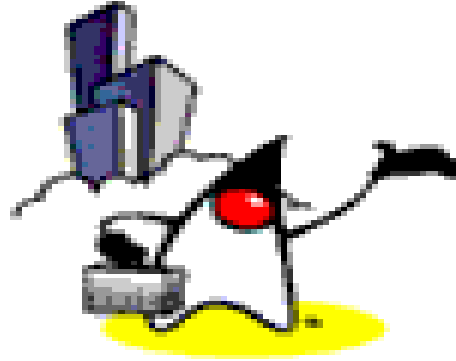
```
// Create content tree from factory object
```

```
com.acme.foo.PurchaseOrder po =  
    com.acme.foo.ObjectFactory.createPurchaseOrder();
```

```
// Once the client application has an instance of the the schema derived  
// object, it can use the mutator methods to set content on it.
```

```
// Set attribute per constraint specified in XML schema
```

```
po.setOrderDate( Calendar.getInstance() );
```



JAXB Runtime Operations Processing: Accessing & Modification

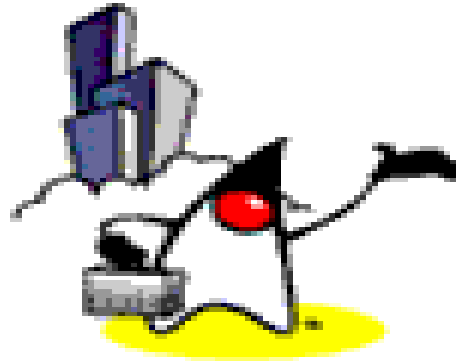
Processing

- Access
- Modify

Example Code: Processing

```
Unmarshaller u = jc.createUnmarshaller();  
PurchaseOrder po =  
    (PurchaseOrder)u.unmarshal( new FileInputStream( "po.xml" ) );
```

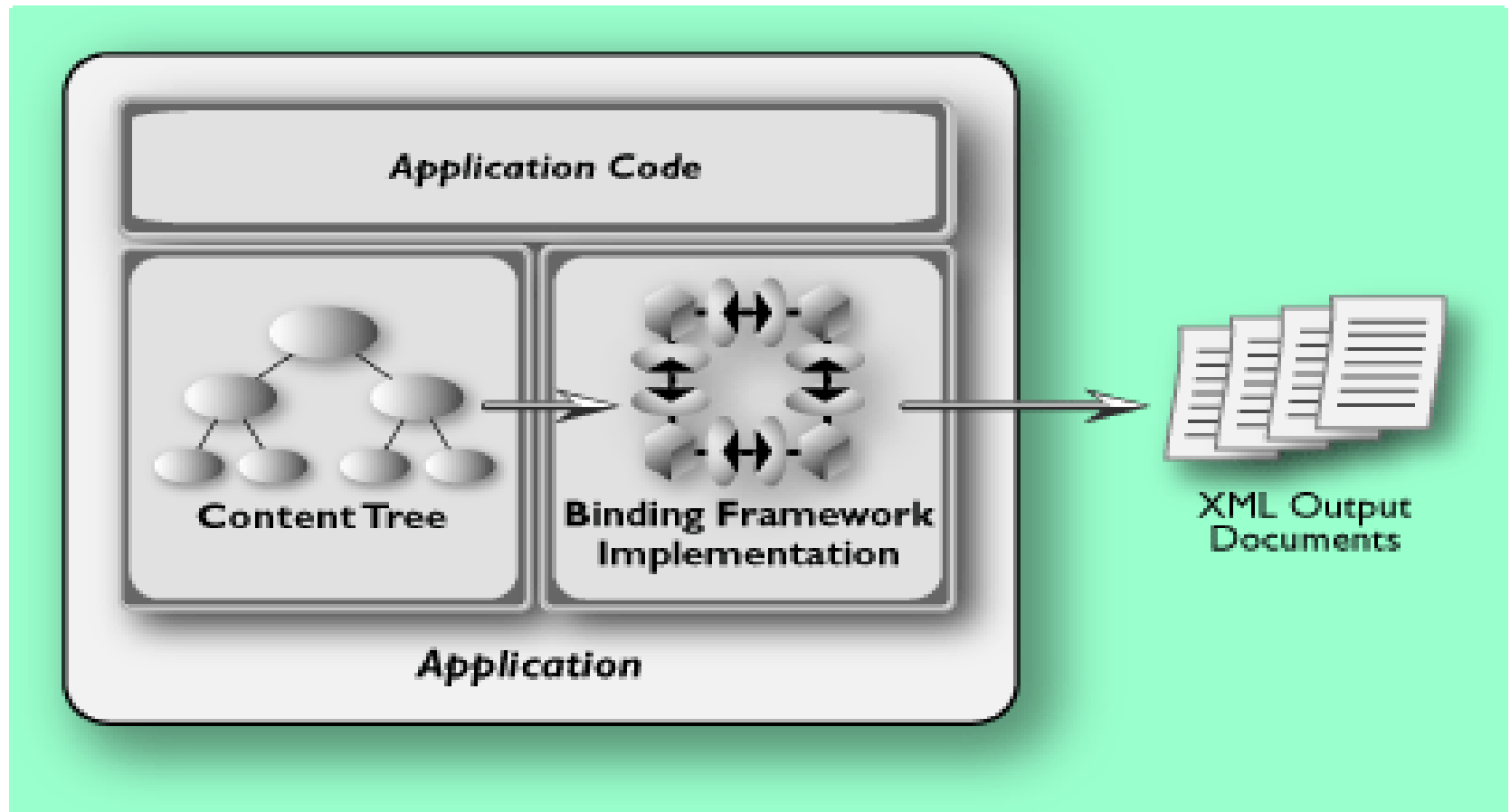
```
USAddress address = po.getBillTo();  
address.setName( "John Bob" );  
address.setStreet( "242 Main Street" );  
address.setCity( "Beverly Hills" );  
address.setState( "CA" );  
address.setZip( new BigDecimal( "90210" ) );
```



JAXB Runtime Operations

Marshalling

Marshalling



Binding Runtime: Marshalling

- Content tree may be marshalled by passing it to `marshal` method of `Marshaller` object
- Content trees are no longer required to be valid before marshalling
 - User discretion on whether validation desirable
 - Marshalling ambiguities handled in an implementation specific manner

javax.xml.bind.Marshaller

- Interface
- Governs the process of serializing Java content trees back into XML data

Marshalling to a File

```
JAXBContext jc = JAXBContext.newInstance(
    "com.acme.foo" );
Unmarshaller u = jc.createUnmarshaller();
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"
    ) );
Marshaller m = jc.createMarshaller();

OutputStream os = new FileOutputStream( "nosferatu.xml" );
m.marshal( obj, os );
```

Marshalling to a SAX Content Handler

```
JAXBContext jc = JAXBContext.newInstance(
    "com.acme.foo" );
Unmarshaller u = jc.createUnmarshaller();
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"
    ) );
Marshaller m = jc.createMarshaller();

// assume MyContentHandler instanceof ContentHandler
m.marshal( obj, new MyContentHandler() );
```

Marshalling to a DOM Node

```
JAXBContext jc = JAXBContext.newInstance(
    "com.acme.foo" );
Unmarshaller u = jc.createUnmarshaller();
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"
    ) );
Marshaller m = jc.createMarshaller();

DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.newDocument();
m.marshal( obj, doc );
```

Marshalling to a `java.io.OutputStream`

```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );
```

```
Unmarshaller u = jc.createUnmarshaller();
```

```
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"  
    ) );
```

```
Marshaller m = jc.createMarshaller();
```

```
m.marshal( obj, System.out );
```

Marshalling to a java.io.Writer

```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );  
Unmarshaller u = jc.createUnmarshaller();  
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"  
    ) );  
Marshaller m = jc.createMarshaller();  
  
m.marshal( obj, new PrintWriter( System.out ) );
```

Marshalling to a `javax.xml.transform.SAXResult`

```
JAXBContext jc = JAXBContext.newInstance(
    "com.acme.foo" );
Unmarshaller u = jc.createUnmarshaller();
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"
    ) );
Marshaller m = jc.createMarshaller();

// assume MyContentHandler instanceof ContentHandler
SAXResult result = new SAXResult( new
    MyContentHandler() );
m.marshal( obj, result );
```

Marshalling to a `javax.xml.transform.DOMResult`

```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );
```

```
Unmarshaller u = jc.createUnmarshaller();
```

```
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"  
    ) );
```

```
Marshaller m = jc.createMarshaller();
```

```
DOMResult result = new DOMResult();
```

```
m.marshal( obj, result );
```


Marshalling to a `javax.xml.transform.StreamResult`

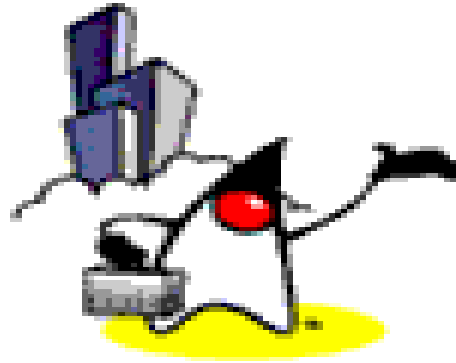
```
JAXBContext jc = JAXBContext.newInstance(  
    "com.acme.foo" );
```

```
Unmarshaller u = jc.createUnmarshaller();
```

```
FooObject obj = (FooObject)u.unmarshal( new File( "foo.xml"  
    ) );
```

```
Marshaller m = jc.createMarshaller();
```

```
StreamResult result = new StreamResult( System.out );  
m.marshal( obj, result );
```



JAXB Runtime Operations Validation

Validation Goals

- Provide seamless access to XML validation in the Java programming environment
- Make convenient to use in the Java environment
 - Relate validation error to instance involved, not file name and line number
- Process as many errors as possible in one validation step
 - Early Access release terminated validation with an exception when first error encountered

Varieties of Validation

- A **type constraint** imposes requirements upon the values that may be given to
 - Attributes
 - Simple type constraint facets in XML Schema
- A **local structural constraint** imposes requirements on every instance of a given element type
- A **global structural constraint** imposes requirements on an entire document

3 Forms of Validation

- Fail-fast Validation
 - Simple runtime type constraint check that can be performed by Property setter method
 - Example:
Bounds check that an integer is between 1 and 10
- On-Demand Validation
 - All 3 types of validation performed on the tree
 - Applications can call the `Validator.validate` method on the Java content tree (or any subtree of it)
- Validation during Unmarshalling

javax.xml.bind.Validator

- Interface
- Controls the validation of content trees during runtime
- Responsible for On-demand validation

Code Example

```
import javax.xml.bind.*;
import generated.packageName.*;

{
    JaxbContext jaxbCtx =
        JaxbContext.newInstance("generated.packageName");
    Unmarshaller um = jaxbCtx.createUnmarshaller();
    Trade trade = (Trade)um.unmarshal(<inputStream>);

    String symbol = trade.getSymbol();
    float price = trade.getPrice();
    if (symbol.equals("WIDGETS") && price > 10.00){
        trade.setQuantity(30);
    }

    Validator validator = jaxbCtx.createValidator();
    validator.validate(trade);

    Marshaller m = jaxbCtx.createMarshaller();
    m.marshal(<outputStream>, trade);
}
```



JAXB Roadmap, Summary, Resources

JAXB Releases

- JAXB 1.0 in March 2003
- JAXB 1.0.1 in JWS DP 1.2 released June 2003
- JAXB 1.0.2 in JWS DP 1.3 released Oct. 2003
- JAXB included in J2EE 1.4 SDK
 - JAXB itself is not part of J2EE 1.4 standard
 - It will be expected to be part of J2SE 1.5

JAXB 1.0.1

- JAXB 1.0.1 - part of Java WSDP 1.2
- Major enhancements for 1.0.1 include:
 - Supports element substitution
 - Bind unsupported XSD concepts using `<xjc:dom>` customization
 - Integrated catalog resolver
 - Enhanced support for extending a schema-derived class
- “Extra Credit” schema support
 - Experimental support for RelaxNG and DTD

JAXB 1.0.2

- JAXB 1.0.2 - part of Java WSDP 1.3
- Major enhancements for 1.0.2 include:
 - XML Schema type substitution

Future: JAXB 2.0

- New JSR 222 submitted:
 - <http://jcp.org/en/jsr/detail?id=222>
- Feature List
 - Ease of development
 - continuation of JAXB 1.0 theme
 - Support all of XML Schema 1.0
 - type substitution, attribute wildcard, ...
 - Java source code to XML schema binding

Future: JAXB 2.0

- Feature List continued
 - JAX-RPC and JAXB Alignment
 - JAX-RPC 2.0 will leverage JAXB 2.0 databinding
 - Association of application behavior with schema-derived code
 - Investigate Design Patterns

Future: JAXB 2.0

Generate J2SE 1.5 Language Extensions

- Builtin Typesafe Enum construct
 - More functionality than design pattern
- Generics and Autoboxing
 - Typesafety for homogenous lists
- JSR 175: Java Source Code Annotations
 - e.g Customize binding of existing JavaBean source code to XML Schema

Leveraging Generics and Autoboxing

```
<xs:complexType name="Student">
  <xs:sequence>
    <xs:element name="grades" type="xs:int"
      maxOccurs="unbounded" />
    ...
  </xs:sequence>
</xs:complexType>
```

Generated by JAXB 2.0

```
Public interface Student {

  /** List of Integer. */
  List<Integer> getGrades();
  ...
}
```

Summary

- A brief introduction to XML Data Binding concepts and terminology
 - Schema-derived Interfaces and Classes
 - Default binding and customizable binding
 - Unmarshalling, Marshalling, Validation
- XML and Java™ technology go together
 - Portable data + portable code
- The Java API for XML Binding (JAXB) is the glue

Resources

- Join the public JAXB interest list
 - <http://jaxb.dev.java.net>
- Download Java WSDP 1.3 containing JAXB 1.0.2:
 - <http://java.sun.com/webservices/download.html>
- “Why Data Binding Matters” article in onjava.com by Brett McLaughlin (05/2002)
 - www.onjava.com/pub/a/onjava/2002/05/15/databind.html
- “XML and Java technologies” article written by Dennis M. Sosnoski (01/2003)
 - www-106.ibm.com/developerworks/library/x-databdopt/



JAXB

(JavaTM Architecture for XML Binding)

