

XML & Java

XML

- Stands for Extensible Markup Language
- Lets you encode complex data in a form that the recipient can parse easily
- Is independent from any programming language

Advantages of XML

- Example: encode product descriptions to be transferred to another computer

- Naïve encoding:

Toaster
29.95

- XML encoding of the same data:

```
<product>  
  <description>Toaster</description>  
  <price>29.95</price>  
</product>
```

Advantages of XML

- XML files are readable by both computers and humans
- XML formatted data is resilient to change
 - *It is easy to add new data elements*
 - *Old programs can process the old information in the new data format*
- In the naïve format a program might think the new data element is the name of the product:

Toaster

29.95

General Appliances

- When using XML it is easy to add new elements:

```
<product>
  <description>Toaster</description>
  <price>29.95</price>
  <manufacturer>General Appliances</manufacturer>
</product>
```

Similarities between XML and HTML

- Both use tags
- Tags are enclosed in angle brackets
- A start-tag is paired with an end-tag that starts with a slash / character
- HTML example:

```
<li>A list item</li>
```

- XML example:

```
<price>29.95</price>
```

Differences Between XML and HTML

- XML tags are case-sensitive
 - * is different from *
- Every XML start-tag must have a matching end-tag
- If a tag has no end-tag, it must end in />

``

- XML attribute values must be enclosed in quotes

``

Differences Between XML and HTML

- HTML describes web documents
- XML can be used to specify many different kinds of data
 - *VRML uses XML syntax to describe virtual reality scenes*
 - *MathML uses XML syntax to describe mathematical formulas*
 - *You can use the XML syntax to describe your own data*
- XML does not tell you how to display data;
it is a convenient format for representing data

The Structure of an XML Document

- An XML data set is called a document

- The document starts with a header

```
<?xml version="1.0"?>
```

- The data are contained in a root element

```
<?xml version="1.0"?>
```

```
<invoice>
```

```
    more data
```

```
</invoice>
```

- The document contains elements and text

The Structure of an XML Document

- An XML element has one of two forms

`<elementName> content </elementName>`

or

`<elementName/>`

- The *contents* can be elements or text or both

The Structure of an XML Document

- An example of an element with both elements and text (mixed content):

```
<p>Use XML for <strong>robust</strong> data formats.</p>
```

- The p element contains
 1. *The text: "Use XML for "*
 2. *A strong child element*
 3. *More text: " data formats."*
- Avoid mixed content for data descriptions (e.g. our product data)
- Content that consists only of elements is called *element content*

The Structure of an XML Document

- An element can have attributes
- The a element in HTML has an href attribute
`<a href="http://java.sun.com"> ... </a>`
- An attribute has a name (such as href) and a value
- The attribute value is enclosed in single or double quotes
- An element can have multiple attributes
``
- An element can have both attributes and content
`<a href="http://java.sun.com">Sun's Java web site</a>`

The Structure of an XML Document

- Attribute is intended to provide information *about* the element content

- Bad use of attributes:

```
<product description="Toaster" price="29.95"/>
```

- Good use of attributes:

```
<product>
  <description>Toaster</description>
  <price currency="USD">29.95</price>
</product>
```

- In this case, the currency attribute helps interpret the element content: `<price currency="EUR">29.95</price>`

Self Check 1

Write XML code with a student element and child elements name and id that describe you.

Answer:

```
<student>  
  <name>James Bond</name>  
  <id>007</id>  
</student>
```

Self Check 2

Why does HTML use the `src` attribute to specify the source of an image instead of `<img>hamster.jpeg</img>`?

Answer: The text `hamster.jpg` is never displayed, so it should not be a part of the document. Instead, the `src` attribute tells the browser where to find the image that should be displayed.

Parsing XML Documents

- A parser is a program that
 - *Reads a document*
 - *Checks whether it is syntactically correct*
 - *Takes some action as it processes the document*
- There are two kinds of XML parsers
 - *SAX (Simple API to XML)*
 - *DOM (Document Object Model)*

Parsing XML Documents

- A parser is a program that
 - *Reads a document*
 - *Checks whether it is syntactically correct*
 - *Takes some action as it processes the document*
- There are two kinds of XML parsers
 - *SAX (Simple API to XML)*
 - *DOM (Document Object Model)*

Parsing XML Documents

- SAX parser (SAXParserFactory, SAXParser, ...)
 - *Event-driven*
 - *It calls a method you provide to process each construct it encounters*
 - *More efficient for handling large XML documents*
 - *Gives you the information in bits and pieces*
- DOM parser (uses the SAX parser)
 - *Builds a tree that represents the document*
 - *When the parser is done, you can analyze the tree*
 - *Easier to use for most applications*
 - *Parse tree gives you a complete overview of the data*
 - *DOM standard defines interfaces and methods to analyze and modify the tree structure that represents an XML document*

JAXP

- Stands for *Java API for XML Processing*
- For creating, reading, and writing XML documents
- Specification defined by Sun Microsystems
- Provides a standard mechanism for DOM parsers to read and create documents

Parsing XML Documents

- Document interface describes the tree structure of an XML document
- A `DocumentBuilder` can generate an object of a class that implements Document interface
- Get a `DocumentBuilder` by calling the static `newInstance` method of `DocumentBuilderFactory`
- Call `newDocumentBuilder` method of the factory to get a `DocumentBuilder`

```
DocumentBuilderFactory factory =  
DocumentBuilderFactory.newInstance(); DocumentBuilder  
builder = factory.newDocumentBuilder();
```

Parsing XML Documents

- To read a document from a file

```
String fileName = . . . ;  
File f = new File(fileName);  
Document doc = builder.parse(f);
```

- To read a document from a URL on the Internet

```
String urlName = . . . ;  
URL u = new URL(urlName);  
Document doc = builder.parse(u);
```

- To read from an input stream

```
InputStream in = . . . ;  
Document doc = builder.parse(in);
```

Parsing XML Documents

- You can inspect or modify the document
- Easiest way of inspecting a document is *XPath* syntax
- An XPath describes a node or set of nodes
- XPath uses a syntax similar to directory paths

An XML Document

```
<?xml version="1.0"?>
<items>
  <item>
    <product>
      <description>Ink Jet Refill Kit</description>
      <price>29.95</price>
    </product>
    <quantity>8</quantity>
  </item>
  <item>
    <product>
      <description>4-port Mini Hub</description>
      <price>19.95</price>
    </product>
    <quantity>4</quantity>
  </item>
</items>
```

Figure 3 An XML Document

The Tree View of XML Document

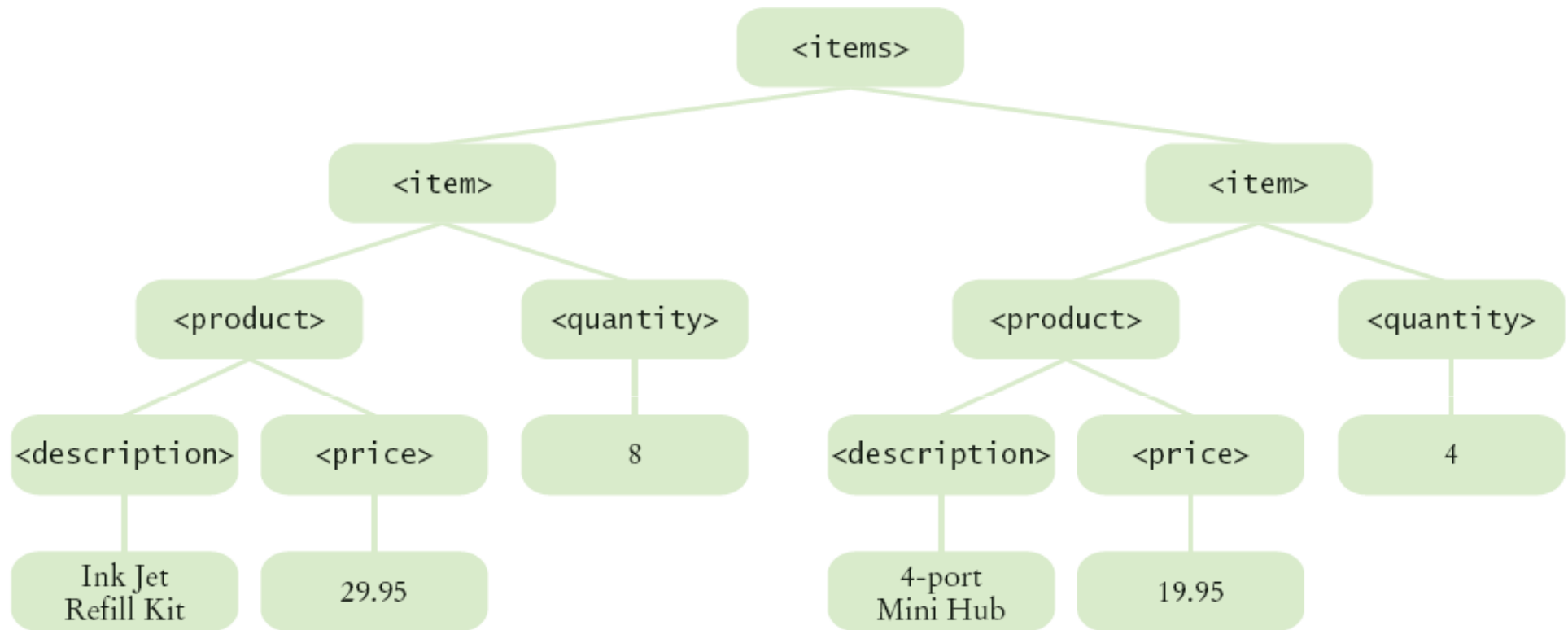


Figure 4 The Tree View of the Document

Parsing XML Documents

- Consider the following XPath, applied to the document in Figure 4:

`/items/item[1]/quantity`

it selects the quantity of the first item (the value 8)

- In XPath, array positions start with 1
- Similarly, you can get the price of the second product as

`/items/item[2]/product/price`

XPath Syntax Summary

Syntax Element	Purpose	Example
<i>name</i>	Matches an element	Item
/	Separates elements	/items/item
[<i>n</i>]	Selects a value from a set	/items/item[1]
@ <i>name</i>	Matches an attribute	Price/@currency
*	Matches anything	/items/*[1]
Count	Counts matches	Count(/items/item)
name	The name of a match	Name(/items/*[1])

Parsing XML Documents

- To get the number of items (2), use the XPath expression:

```
count(/items/item)
```

- The total number of children (2) can be obtained as:

```
count(/items/*)
```

- To select attributes, use an @ followed by the name of the attribute:

```
/items/item[2]/product/price/@currency
```

- To find out the name of a child in a document with variable/unknown structure:

```
name(/items/item[1]/*[1])
```

The result is the name of the first child of the first item, or product

Parsing XML Documents

- To evaluate an XPath expression in Java, create an XPath object

```
XPathFactory xpfactory = XPathFactory.newInstance();  
XPath path = xpfactory.newXPath();
```

- Then call the evaluate method

```
String result = path.evaluate(expression, doc);
```

- *expression* is an XPath expression
- *doc* is the Document object that represents the XML document

- For example,

```
String result = path.evaluate("/items/item[2]/  
    product/price", doc);
```

sets result to the string "19.95".

Parsing XML Documents: An Example

- ItemListParser parses an XML document with a list of product descriptions
 - *Uses the LineItem and Product*

- parse takes the file name and returns an array list of LineItem objects:

```
ItemListParser parser = new ItemListParser();  
ArrayList<LineItem> items = parser.parse("items.xml");
```

- ItemListParser translates each XML element into an object of the corresponding Java class

Parsing XML Documents: An Example

- We first get the number of items:

```
int itemCount = Integer.parseInt(path.evaluate(
    "count(/items/item)", doc));
```

- For each item element, we gather the product data and construct a Product object:

```
String description = path.evaluate(
    "/items/item[" + i + "]/product/description", doc);
double price = Double.parseDouble(path.evaluate(
    "/items/item[" + i + "]/product/price", doc));
Product pr = new Product(description, price);
```

- Then we construct a LineItem object, and add it to the items array list

ItemListParser.java

```
01: import java.io.File;
02: import java.io.IOException;
03: import java.util.ArrayList;
04: import javax.xml.parsers.DocumentBuilder;
05: import javax.xml.parsers.DocumentBuilderFactory;
06: import javax.xml.parsers.ParserConfigurationException;
07: import javax.xml.xpath.XPath;
08: import javax.xml.xpath.XPathExpressionException;
09: import javax.xml.xpath.XPathFactory;
10: import org.w3c.dom.Document;
11: import org.xml.sax.SAXException;
12:
13: /**
14:     An XML parser for item lists
15: */
16: public class ItemListParser
17: {
18:     /**
19:         Constructs a parser that can parse item lists.
20:     */
```

Continued

ItemListParser.java (cont.)

```
21:     public ItemListParser()
22:         throws ParserConfigurationException
23:     {
24:         DocumentBuilderFactory dbfactory
25:             = DocumentBuilderFactory.newInstance();
26:         builder = dbfactory.newDocumentBuilder();
27:         XPathFactory xpfactory = XPathFactory.newInstance();
28:         path = xpfactory.newXPath();
29:     }
30:
31:     /**
32:      * Parses an XML file containing an item list.
33:      * @param fileName the name of the file
34:      * @return an array list containing all items in the XML file
35:      */
36:     public ArrayList<LineItem> parse(String fileName)
37:         throws SAXException, IOException, XPathExpressionException
38:     {
39:         File f = new File(fileName);
40:         Document doc = builder.parse(f);
41:
```

Continued

ItemListParser.java (cont.)

```
42:     ArrayList<LineItem> items = new ArrayList<LineItem>();
43:     int itemCount = Integer.parseInt(path.evaluate(
44:         "count(/items/item)", doc));
45:     for (int i = 1; i <= itemCount; i++)
46:     {
47:         String description = path.evaluate(
48:             "/items/item[" + i + "]/product/description", doc);
49:         double price = Double.parseDouble(path.evaluate(
50:             "/items/item[" + i + "]/product/price", doc));
51:         Product pr = new Product(description, price);
52:         int quantity = Integer.parseInt(path.evaluate(
53:             "/items/item[" + i + "]/quantity", doc));
54:         LineItem it = new LineItem(pr, quantity);
55:         items.add(it);
56:     }
57:     return items;
58: }
59:
60: private DocumentBuilder builder;
61: private XPath path;
62: }
63:
```

Continued

ItemListParserDemo.java

```
01: import java.util.ArrayList;
02:
03: /**
04:     This program parses an XML file containing an item list.
05:     It prints out the items that are described in the XML file.
06: */
07: public class ItemListParserDemo
08: {
09:     public static void main(String[] args) throws Exception
10:     {
11:         ItemListParser parser = new ItemListParser();
12:         ArrayList<LineItem> items = parser.parse("items.xml");
13:         for (LineItem anItem : items)
14:             System.out.println(anItem.format());
15:     }
16: }
```

Continued

ItemListParserDemo.java (cont.)

Output:

```
Ink Jet Refill Kit 29.95 8 239.  
6 4-port Mini Hub 19.95 4 79.8
```

Self Check 4

What is the result of evaluating the XPath statement `/items/item[1]/product/price` in the XML document of Figure 4?

Answer: 29.95.

Self Check 5

Which XPath statement yields the name of the root element of any XML document?

Answer: `name (/ * [1] ) .`

Grammars, Parsers, and Compilers

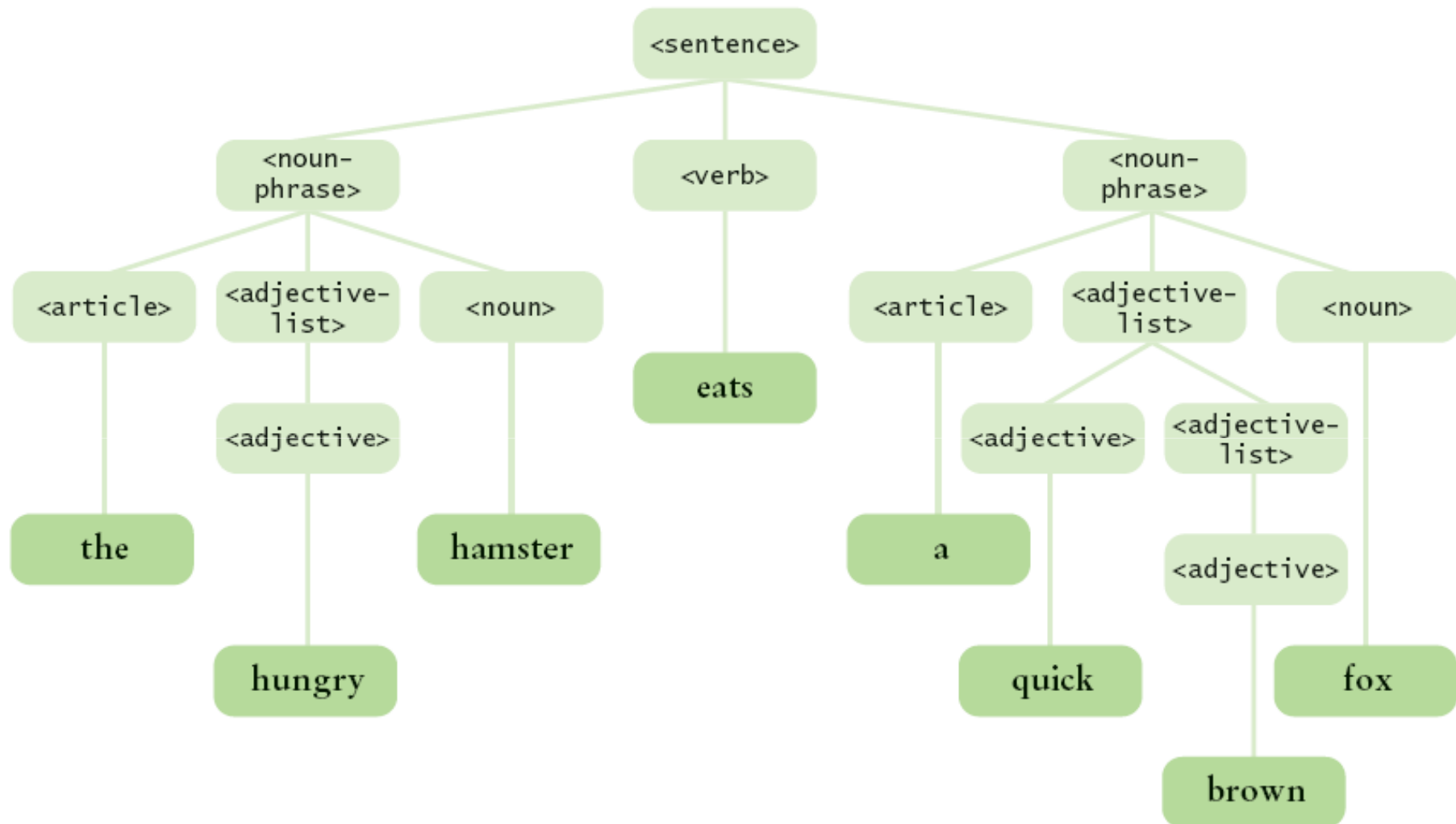


Figure 5 A Parse Tree for a Simple Sentence

Creating XML Documents

- We can build a Document object in a Java program and then save it as an XML document
- We need a DocumentBuilder object to create a new, empty document

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();  
Document doc = builder.newDocument(); // An empty  
    document
```

- The Document class has methods to create elements and text nodes

Creating XML Documents

- To create an element use `createElement` method and pass it a tag
`Element priceElement = doc.createElement("price");`
- Use `setAttribute` method to add an attribute to the tag
`priceElement.setAttribute("currency", "USD");`
- To create a text node, use `createTextNode` and pass it a string
`Text textNode = doc.createTextNode("29.95");`
- Then add the text node to the element:
`priceElement.appendChild(textNode);`

DOM Interfaces for XML Document Nodes

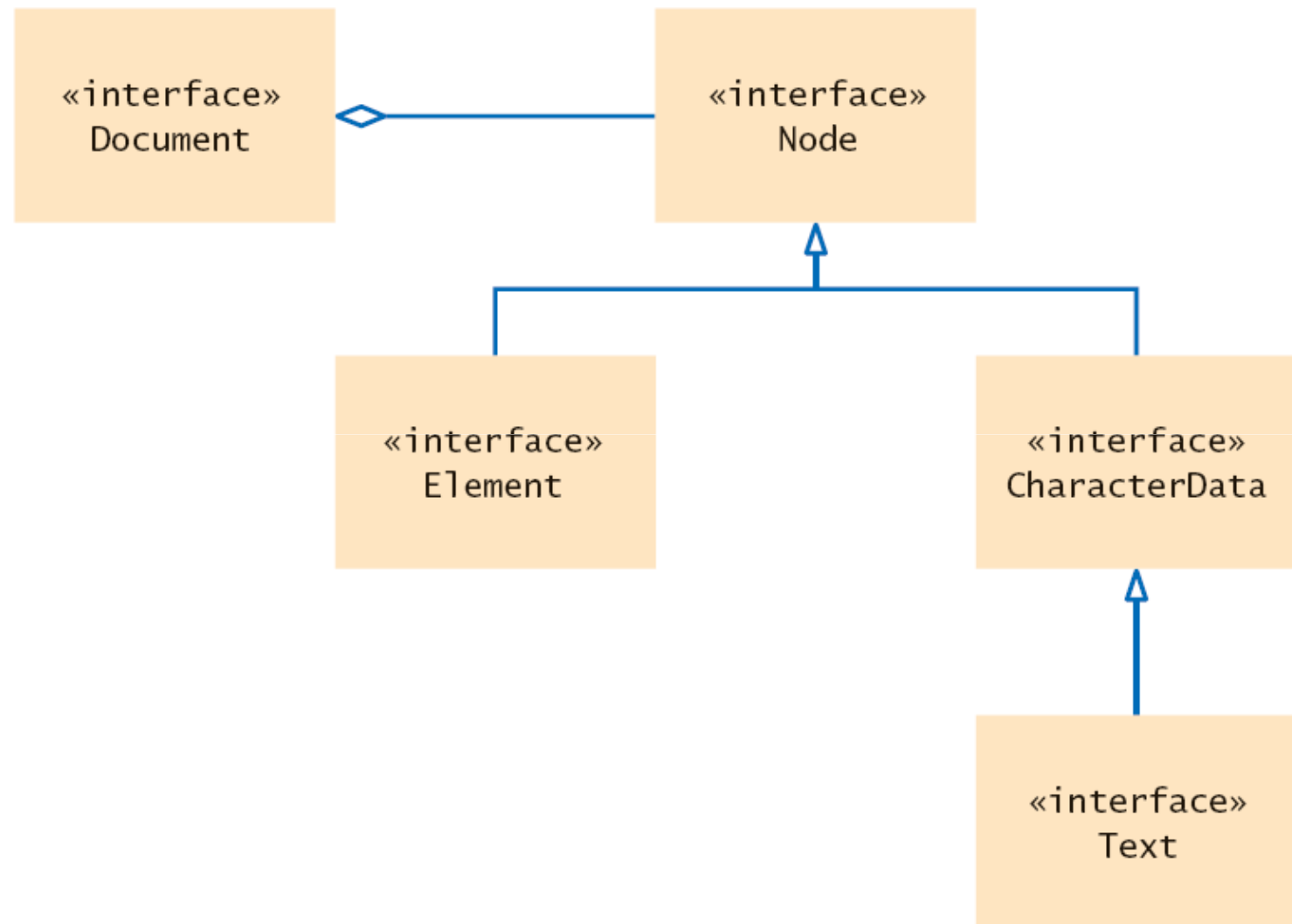


Figure 7 UML Diagram of DOM Interfaces Used in This Chapter

Creating XML Documents

- To construct the tree structure of a document, it is a good idea to use a set of helper methods
- Helper method to create an element with text:

```
private Element createTextElement(String name, String
    text)
{
    Text t = doc.createTextNode(text);
    Element e = doc.createElement(name);
    e.appendChild(t);
    return e;
}
```

- To construct a price element:

```
Element priceElement = createTextElement("price",
    "29.95");
```

Creating XML Documents

- Helper method to create a product element from a Product object:

```
private Element createProduct(Product p)
{
    Element e = doc.createElement("product");
    e.appendChild(createTextElement("description",
        p.getDescription()));
    e.appendChild(createTextElement("price", "" +
        p.getPrice()));
    return e;
}
```

Continued

Creating XML Documents (cont.)

- createProduct is called from createItem:

```
private Element createItem(LineItem anItem)
{
    Element e = doc.createElement("item");
    e.appendChild(createProduct(anItem.getProduct()));
    e.appendChild(createTextElement( "quantity", "" +
        anItem.getQuantity()));
    return e;
}
```

Creating XML Documents

- A helper method

```
private Element createItems(ArrayList<LineItem> items)
```

is implemented in the same way

- Build the document as follows:

```
ArrayList<LineItem> items = . . .;  
doc = builder.newDocument();  
Element root = createItems(items);  
doc.appendChild(root);
```

Creating XML Documents

- There are several ways of writing an XML document
- We use the `LSSerializer` interface
- Obtain an `LSSerializer` with the following magic incantation:

```
DOMImplementation impl = doc.getImplementation();  
DOMImplementationLS implLS  
    = (DOMImplementationLS) impl.getFeature("LS",  
        "3.0");  
LSSerializer ser = implLS.createLSSerializer();
```

- Then you simply use the `writeToString` method:

```
String str = ser.writeToString(doc);
```
- The `LSSerializer` produces an XML document without spaces or line breaks

ItemListBuilder.java

```
01: import java.util.ArrayList;
02: import javax.xml.parsers.DocumentBuilder;
03: import javax.xml.parsers.DocumentBuilderFactory;
04: import javax.xml.parsers.ParserConfigurationException;
05: import org.w3c.dom.Document;
06: import org.w3c.dom.Element;
07: import org.w3c.dom.Text;
08:
09: /**
10:     Builds a DOM document for an array list of items.
11: */
12: public class ItemListBuilder
13: {
14:     /**
15:         Constructs an item list builder.
16:     */
17:     public ItemListBuilder()
18:         throws ParserConfigurationException
19:     {
20:         DocumentBuilderFactory factory
21:             = DocumentBuilderFactory.newInstance();
```

Continued

ItemListBuilder.java (cont.)

```
22:     builder = factory.newDocumentBuilder();
23: }
24:
25: /**
26:     Builds a DOM document for an array list of items.
27:     @param items the items
28:     @return a DOM document describing the items
29: */
30: public Document build(ArrayList<LineItem> items)
31: {
32:     doc = builder.newDocument();
33:     doc.appendChild(createItems(items));
34:     return doc;
35: }
36:
37: /**
38:     Builds a DOM element for an array list of items.
39:     @param items the items
40:     @return a DOM element describing the items
41: */
42: private Element createItems(ArrayList<LineItem> items)
43: {
44:     Element e = doc.createElement("items");
```

Continued

ItemListBuilder.java (cont.)

```
45:
46:     for (LineItem anItem : items)
47:         e.appendChild(createItem(anItem));
48:
49:     return e;
50: }
51:
52: /**
53:     Builds a DOM element for an item.
54:     @param anItem the item
55:     @return a DOM element describing the item
56: */
57: private Element createItem(LineItem anItem)
58: {
59:     Element e = doc.createElement("item");
60:
61:     e.appendChild(createProduct(anItem.getProduct()));
62:     e.appendChild(createTextElement(
63:         "quantity", "" + anItem.getQuantity()));
64:
65:     return e;
66: }
67:
```

Continued

ItemListBuilder.java (cont.)

```
68:  /**
69:      Builds a DOM element for a product.
70:      @param p the product
71:      @return a DOM element describing the product
72:  */
73:  private Element createProduct(Product p)
74:  {
75:      Element e = doc.createElement("product");
76:
77:      e.appendChild(createTextElement(
78:          "description", p.getDescription()));
79:      e.appendChild(createTextElement(
80:          "price", "" + p.getPrice()));
81:
82:      return e;
83:  }
84:
85:  private Element createTextElement(String name, String text)
86:  {
87:      Text t = doc.createTextNode(text);
88:      Element e = doc.createElement(name);
89:      e.appendChild(t);
90:      return e;
```

Continued

ItemListBuilder.java (cont.)

```
91:     }  
92:  
93:     private DocumentBuilder builder;  
94:     private Document doc;  
95: }
```

ItemListBuilderDemo.java

```
01: import java.util.ArrayList;
02: import org.w3c.dom.DOMImplementation;
03: import org.w3c.dom.Document;
04: import org.w3c.dom.ls.DOMImplementationLS;
05: import org.w3c.dom.ls.LSSerializer;
06:
07: /**
08:     This program demonstrates the item list builder. It prints the XML
09:     file corresponding to a DOM document containing a list of items.
10: */
11: public class ItemListBuilderDemo
12: {
13:     public static void main(String[] args) throws Exception
14:     {
15:         ArrayList<LineItem> items = new ArrayList<LineItem>();
16:         items.add(new LineItem(new Product("Toaster", 29.95), 3));
17:         items.add(new LineItem(new Product("Hair dryer", 24.95), 1));
18:
19:         ItemListBuilder builder = new ItemListBuilder();
20:         Document doc = builder.build(items);
```

Continued

ItemListBuilderDemo.java (cont.)

```
21:      DOMImplementation impl = doc.getImplementation();
22:      DOMImplementationLS implLS
23:          = (DOMImplementationLS) impl.getFeature("LS", "3.0");
24:      LSSerializer ser = implLS.createLSSerializer();
25:      String out = ser.writeToString(doc);
26:
27:      System.out.println(out);
28:  }
29: }
```

Continued

ItemListBuilderDemo.java (cont.)

Output:

```
<?xml version="1.0" encoding="UTF-8"?><items><item><
product>
<description>Toaster</description><price>29.95</price><
/product>
<quantity>3</quantity></item><item><product><
description>Hair dryer
</description><price>24.95</price></product><quantity>
1</quantity>
</item></items>
```

Self Check 6

How would you write a document to the file output.xml?

Answer: First construct a string, as described, and then use a `PrintWriter` to save the string to a file.

Validating XML Documents

- We need to specify rules for XML documents of a particular type
- There are several mechanisms for this purpose
- The oldest and simplest mechanism is a Document Type Definition (DTD)

Document Type Definitions

- A DTD is a set of rules for correctly formed documents of a particular type
 - *Describes the valid attributes for each element type*
 - *Describes the valid child elements for each element type*
- Valid child elements are described by an ELEMENT rule

```
<!ELEMENT items (item*)>
```
- The `items` element can have 0 or more `item` elements
- Definition of an `item` node

```
<! ELEMENT item (product, quantity)>
```
- Children of the `item` node must be a `product` node followed by a `quantity` node

Document Type Definitions

- Definition of product node

```
<! ELEMENT product (description, price)>
```

- The other nodes

```
<!ELEMENT quantity (#PCDATA)>
```

```
<!ELEMENT description
```

```
(#PCDATA)> <!ELEMENT price (#PCDATA)>
```

- #PCDATA refers to text, called "parsed character data" in XML terminology
 - *Can contain any characters*
 - *Special characters have to be replaced when they occur in character data*

Replacements for Special Characters

Character	Encoding	Name
<	<	Less than (left angle bracket)
>	>	Greater than (right angle bracket)
&	&	Ampersand
'	'	Apostrophe
"	"	Quotation mark

DTD for Item List

```
<!ELEMENT items (item)*>
<!ELEMENT item (product, quantity)>
<!ELEMENT product (description, price)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Regular Expressions for Element Content

Rule Description	Element Content
EMPTY	No children allowed
(E^*)	Any sequence of 0 or more elements E
(E^+)	Any sequence of 1 or more elements E
($E?$)	Optional element E (0 or 1 occurrences allowed)
($E_1, E_2, \dots$)	Element E_1 , followed by $E_2, \dots$
($E_1 \mid E_2 \mid \dots$)	Element E_1 or E_2 or $\dots$
(#PCDATA)	Text only
(#PCDATA $\mid E_1 \mid$ $\#2 \dots$) [*]	Any sequence of text and elements $E_1, E_2, \dots$, in any order
ANY	Any children allowed

Document Type Definitions

- The HTML DTD defines the `img` element to be EMPTY
 - *An image has only attributes*
- More interesting child rules can be formed with the regular expression operations
(* + ? , |)

DTD Regular Expression Operations

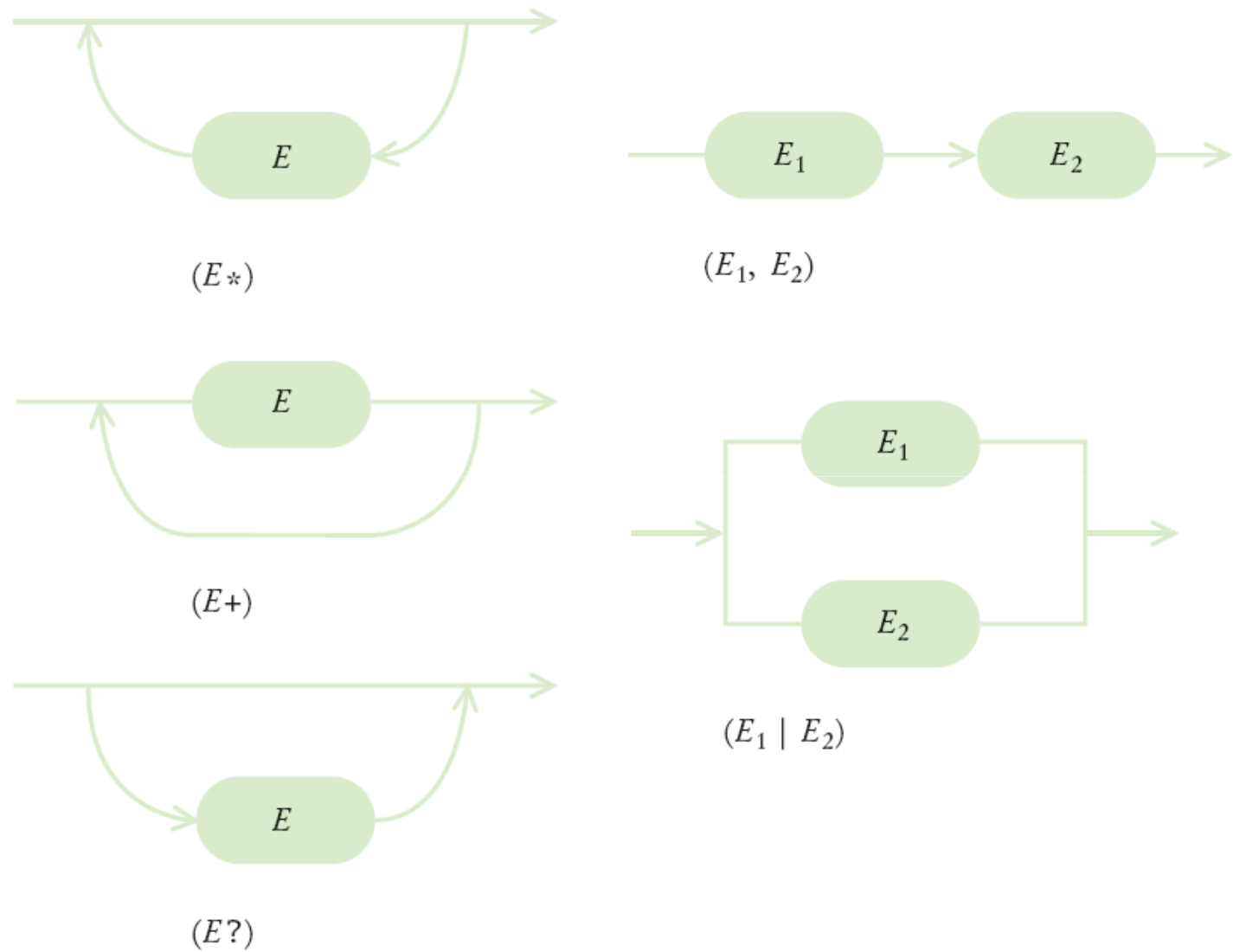


Figure 8 DTD Regular Expression Operations

DTD Regular Expression Operations

- For example,

```
<!ELEMENT section (title, (paragraph | (image,  
title?)))+>
```

defines an element `section` whose children are:

- A *title element*
- A *sequence of one or more of the following*:
 - paragraph elements
 - image elements followed by optional `title` elements

Continued

DTD Regular Expression Operations (cont.)

- Thus, the following is not valid

```
<section>  
  <paragraph/>  
  <paragraph/>  
  <title/>  
</section>
```

because there is no starting title, and the title at the end doesn't follow an image

Document Type Definitions

- A DTD gives you control over the allowed attributes of an element

`<!ATTLIST Element Attribute Type Default>`

- Type can be any sequence of character data specified as CDATA
- There is no practical difference between the CDATA and #PCDATA
- Use CDATA in attribute declarations
- #PCDATA in element declarations
- You can also specify a finite number of choices

`<!ATTLIST price currency (USD | EUR | JPY ) #REQUIRED >`

- You can use letters, numbers, and the characters - _ for the attribute values

Common Attribute Types

Type Description	Attribute Type
CDATA	Any character data
(V1 V2 . . .)	One of $V_1, V_2, \dots$

Attribute Defaults

Default Declaration	Explanation
#REQUIRED	Attribute is required
#IMPLIED	Attribute is optional
V	Default attribute, to be used if attribute is not specified
#FIXED V	Attribute must either be unspecified or contain this value

Document Type Definitions

- #IMPLIED keyword means you can supply an attribute or not.

```
<!ATTLIST price currency CDATA #IMPLIED>
```

- If you omit the attribute, the application processing the XML data implicitly assumes some default value
- You can specify a default to be used if the attribute is not specified

```
<!ATTLIST price currency CDATA "USD">
```

- To state that an attribute can only be identical to a particular value:

```
<!ATTLIST price currency CDATA #FIXED "USD">
```

Specifying a DTD in an XML Document

- An XML document can reference a DTD in one of two ways
 1. *The document may contain the DTD*
 2. *The document may refer to a DTD stored elsewhere*
- A DTD is introduced with the DOCTYPE declaration
- If the document contains its DTD, the declaration looks like this:
`<!DOCTYPE rootElement [ rules ]>`

Example: An Item List

```
<?xml version="1.0"?>
<!DOCTYPE items [

<!ELEMENT items (item*)>
<!ELEMENT item (product, quantity)>
<!ELEMENT product (description, price)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Continued

Example: An Item List (cont.)

```
]>
<items>
  <item>
    <product>
      <description>Ink Jet Refill Kit</description>
      <price>29.95</price>
    </product>
    <quantity>8</quantity>
  </item>
  <item>
    <product>
      <description>4-port Mini Hub</description>
      <price>19.95</price>
    </product>
    <quantity>4</quantity>
  </item>
</items>
```

Specifying a DTD in an XML Document

- If the DTD is more complex, it is better to store it outside the XML document

- *Use the SYSTEM keyword*

```
<!DOCTYPE items SYSTEM "items.dtd" >
```

- The resource might be an URL anywhere on the Web:

```
<!DOCTYPE items SYSTEM  
    "http://www.mycompany.com/dtds/items.dtd">
```

- The DOCTYPE declaration can contain a PUBLIC keyword

```
<!DOCTYPE faces-config PUBLIC  
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces  
        Config 1.0//EN"  
    "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
```

If the public identifier is familiar, the program parsing the document need not spend time retrieving the DTD

Parsing and Validation

- When your XML document has a DTD, you can request validation when parsing
- The parser will check that all child elements and attributes conform to the ELEMENT and ATTLIST rules in the DTD
- The parser reports an error if the document is invalid
- Use the `setValidating` method of the `DocumentBuilderFactory` before calling `newDocumentBuilder` method

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
factory.setValidating(true);  
DocumentBuilder builder = factory.newDocumentBuilder();  
Document doc = builder.parse(. . .);
```

Parsing with Document Type Definitions

- When you parse an XML file with a DTD, tell the parser to ignore white space

```
factory.setValidating(true);  
factory.setIgnoringElementContentWhitespace(true);
```

- If the parser has access to a DTD, it can fill in defaults for attributes
- For example, suppose a DTD defines a currency attribute for a price element:

```
<!ATTLIST price currency CDATA "USD">
```

- If a document contains a price element without a currency attribute, the parser can supply the default:

```
String attributeValue =  
    priceElement.getAttribute("currency");  
    // Gets "USD" if no currency specified
```

Self Check 7

How can a DTD specify that the `quantity` element in an `item` is optional?

Answer:

```
<!ELEMENT item (product, quantity?)>
```

Self Check 8

How can a DTD specify that a product element can contain a description and a price element, in any order?

Answer:

```
<!ELEMENT product ((description, price) | (price,
description))>
```

Self Check 9

How can a DTD specify that the description element has an optional attribute language?

Answer:

```
<!ATTLIST description language CDATA #IMPLIED>
```

DTD Example1

```
<!DOCTYPE TVSCHEDULE [  
    <!ELEMENT TVSCHEDULE (CHANNEL+)>  
    <!ELEMENT CHANNEL (BANNER,DAY+)>  
    <!ELEMENT BANNER (#PCDATA)>  
    <!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>  
    <!ELEMENT HOLIDAY (#PCDATA)>  
    <!ELEMENT DATE (#PCDATA)>  
    <!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>  
    <!ELEMENT TIME (#PCDATA)>  
    <!ELEMENT TITLE (#PCDATA)>  
    <!ELEMENT DESCRIPTION (#PCDATA)>  
  
    <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  
    <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  
    <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  
    <!ATTLIST TITLE RATING CDATA #IMPLIED>  
    <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  

```

DTD Example 2

```
<!DOCTYPE NEWSPAPER [  
    <!ELEMENT NEWSPAPER (ARTICLE+)>  
    <!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  
    <!ELEMENT HEADLINE (#PCDATA)>  
    <!ELEMENT BYLINE (#PCDATA)>  
    <!ELEMENT LEAD (#PCDATA)>  
    <!ELEMENT BODY (#PCDATA)>  
    <!ELEMENT NOTES (#PCDATA)>  
  
    <!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  
    <!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  
    <!ATTLIST ARTICLE DATE CDATA #IMPLIED>  
    <!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  
  
    <!ENTITY NEWSPAPER "Vervet Logic Times">  
    <!ENTITY PUBLISHER "Vervet Logic Press">  
    <!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">  
  

```