Welcome to:

DOM Parser

© Copyright IBM Corporation 2004
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit Objectives

After completing this unit, you should be able to:

- List the reasons you would use DOM
- Describe the Abstract Model for DOM
- Describe Java Bindings for DOM
- List and define the best practices for using DOM

© Copyright IBM Corporation 2004

DOM Basics

- Document Object Model
  - API that defines the objects present in an XML document and the methods and properties that are used to define them.
- DOM parser reads and loads an entire XML document into memory as a tree structure.
- DOM stresses an Object Model over a Data Model.
  - Objects protect their data using accessors methods.
  - Data models are centered around the data.
- Every object in the DOM API is considered a node.
  - Data is encapsulated within the node.
  - Special node subtypes for each data type.

© Copyright IBM Corporation 2004

DOM Specifications

- Standard set of programming interfaces.
  - Platform-and language-independent.
  - API defines interface, not the implementation.
  - Parsers provide the actual implementation.
- DOM specification is defined in OMG's IDL Language.
  - A language-independent specification.
  - Looks like a cross between C and Java.
- Defines the semantics of how an XML document is accessed and manipulated.
  - Objects define behavior, attributes, and relationships
  - Provides methods for:
    - Navigation (traverse up, down, sideways through tree)
    - Modification (create, read, write, and delete nodes)

© Copyright IBM Corporation 2004

History of DOM

- DOM originated from JavaScript to represent HTML
  - Dynamic HTML is direct ancestor
  - Formalized for use in XML
- DOM Level 1
  - W3C recommendation - October, 1998
  - Defines Core Interface support for XML and HTML
- DOM Level 2
  - W3C recommendation - November, 2000
  - Added support for: Namespaces, Cascading Style Sheets, and Events
- DOM Level 3
  - W3C recommendation - January 27, 2004
  - Enhanced support for: Namespaces, XPath addressing, abstract DTD and Schema support, and new read/write support

© Copyright IBM Corporation 2004

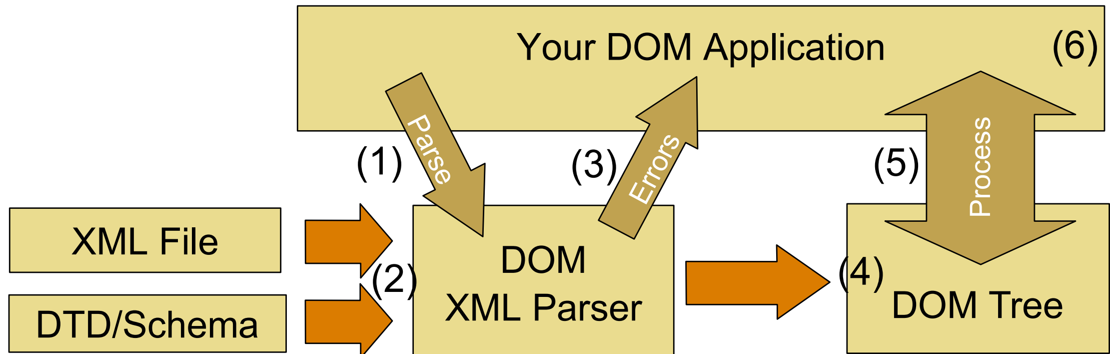
When to Use DOM

- When you need to modify the XML document's
  - Content
  - Structure
  - Order
- When you need to go over a document multiple times
  - No need to rescan the document on each pass
- When you need to merge several XML documents together
  - Consisting of the same structure
  - Consisting of different structure
- Any time you need to do a lot with the document's content
  - Sharing content with other processes
  - Frequent access of content

© Copyright IBM Corporation 2004

DOM Processing Overview

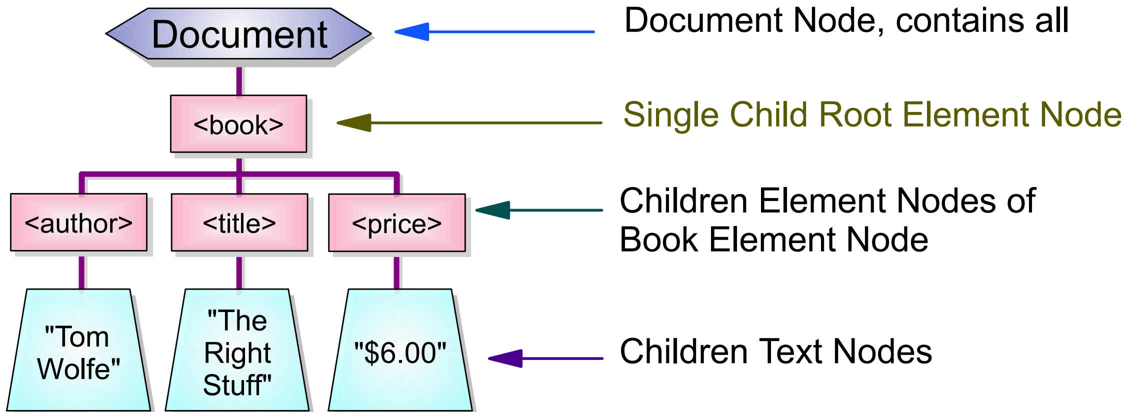
- Basic DOM Processing Flow
  1. Application creates a DOM Parser
  2. Parser reads in XML and any vocabulary
  3. Parser returns any errors to the application
  4. Parser generates a DOM Tree
  5. Application can read, write, or update the DOM Tree
  6. Application writes final XML to output



© Copyright IBM Corporation 2004

DOM Abstract Tree Model

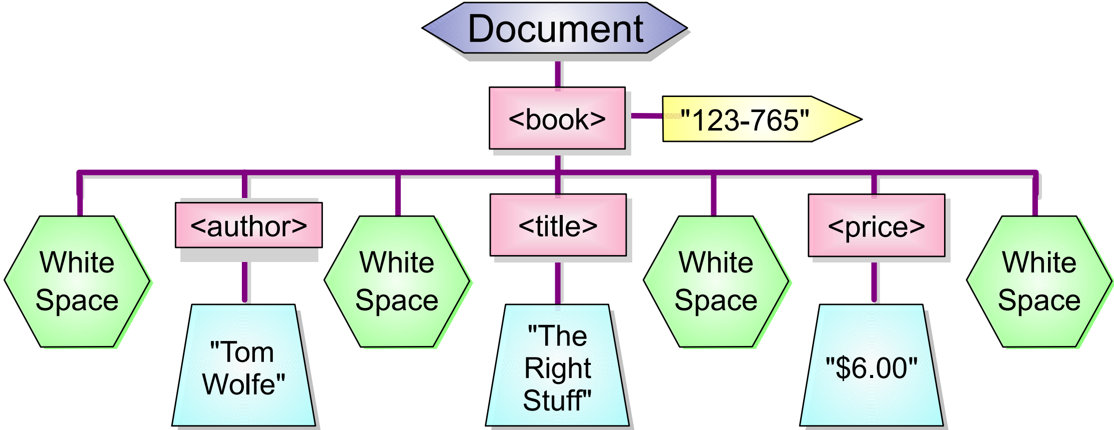
```
<book>
  <author>Tom Wolfe</author>
  <title>The Right Stuff</title>
  <price>$6.00</price>
</book>
```



© Copyright IBM Corporation 2004

Real DOM Tree Structure

```
<book isbn="123-765">ϖ
ϖϖ<author>Tom Wolfe</author>ϖ
ϖϖ<title>The Right Stuff</title>ϖ
ϖϖ<price>$6.00</price>ϖ
</book>
```



© Copyright IBM Corporation 2004

Node Contents and Values for DOM Tree

- Each node in a DOM tree contains attributes describing names, values, and parent/child/sibling relationships.

Node Type	Name / Value	Parent	Children
Document	#document -Base Node of entire document	(before root)	book
Element	**book** - Root element	#document	7 Children 3 Elements; 4 Text (all whitespace)
Attr	**isbn /** "123-765" - Held inside "book" element	N/A	N/A
Text	#text - whitespace	book	null
Element	**author**	book	#text
Text	#text - "Tom Wolfe"	author	null
Text	#text - whitespace	book	null
Element	**title**	book	#text
Text	#text - "The Right Stuff"	title	null
Text	#text - whitespace	book	null
Element	**price**	book	#text
Text	#text - "$6.00"	price	null
Text	#text - whitespace	book	null

© Copyright IBM Corporation 2004

Node Interface

- Everything is a node in the DOM.
- Contains attributes and method for determining:
  - Node name
  - Type of node (element, text, attribute, and so forth)
  - Determining relationships (parent, siblings, children)
- Has methods for modifying structure:
  - Adding children (before or after current location)
  - Replacing children
  - Deleting children
- The top-level node is called the DocumentElement Node:
  - Represents the entire document.
  - Only a single element child node is allowed ("root").
  - May have other children nodes (comments, PIs, and so forth).
- A **Node Interface Model** is used to access the individual elements in the node tree.

© Copyright IBM Corporation 2004

Node Types

- The following are some common node types.

Node	Description
Document	Represents the entire document tree, referenced from the root node.
Element	Represents a common element node within the document tree. May contain attribute nodes.
Attr	Special Attribute Node type that acts as a property (not a child) to an Element Node.
Text	Represents character data children of elements, or attributes, or whitespace between elements.

- And some less common node types.

Node	Description
DocumentFragment	Represents a portion of a document. Not necessarily well-formed.
CDATASection	Represents a CDATA section of unparsed text.
Comment	Represents a comment.
Processing Instruction	Represents a PI for application specific rules.
Entity EntityReference	Represents a parsed or Unparsed Entity. An EntityReference may be used when updating doc.

© Copyright IBM Corporation 2004

Document and DocumentFragment

- Document
  - Represents the entire document
  - Reference point is the root of document tree
  - Primary access point to the data within the tree
- DocumentFragment
  - Represents a partial document
  - Not necessarily well-formed
  - Useful for:
    - Rearranging the DOM tree
    - Modifying the DOM structure
    - Extracting sections from XML
    - Merging multiple pieces together

© Copyright IBM Corporation 2004

DOM Application Structure

- Parser creation
  - Define factory
  - Configure parser settings
  - Create parser
  - Parse document
- Dealing with parsing errors
  - Exceptions
  - SAX-style error handling
- Navigating the document
  - Traversing document tree
- Processing the document
  - Modifying contents
  - Modifying structure
  - Moving nodes

© Copyright IBM Corporation 2004

DOM Sample

- The next charts can be better understood in light of an executable example demonstrated using *Studio*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: ShowDOMInfo.java (executable)
- Input file: data/books.xml Output: -v

```
<!DOCTYPE books SYSTEM "books.dtd" >
<books>
    <book>
        <author>Tom Wolfe</author>
        <title>The Right Stuff</title>
        <price>$6.00</price>
    </book>
    <book>
        <author>R.L. Stevenson</author>
        <title>Treasure Island</title>
        <price>$13.00</price>
    </book>
    <book>
        <author>Carl Hiaasen</author>
        <title>Tourist Season</title>
        <price>$5.99</price>
    </book>
    <book>
        <author>Dave Barry</author>
        <title>Big Trouble</title>
        <price>$3.95</price>
    </book>
    <!-- Plus many, many more books -->
</books>
```

```
Current DocumentBuilderFactory Settings:
     Coalescing: False
     Expands Entity References: True
     Ignoring Comments: False
     Ignoring Whitespace: False
     Namespace Aware: False
     Validating: False
Current DocumentBuilder Settings:
     Namespace Aware: True
     Validating: False
     Style Sheets supported: False
DTD Name = books
DTD System ID = books.dtd
DTD Public ID = null
DTD Name = books
DTD Entities:
DTD Notations:
```

© Copyright IBM Corporation 2004

Parser Creation: Four Steps

- Step 1: Create new instance of DocumentBuilderFactory.
  - JAXP dynamically locates the real DOM implementation.
  - Application doesn't care what DOM parser is used.

```
DocumentBuilderFactory docFactory =
   DocumentBuilderFactory.newInstance();
```

- Step 2: Set any desired parsing options.
  - Some behaviors of the parser can be altered.
  - All available features and properties are shown later.

```
docFactory.setValidating(false);
```

- Step 3: Create a DocumentBuilder from the factory.
  - Multiple parsers can be generated from a single factory.

```
DocumentBuilder builder = docFactory.newDocumentBuilder();
```

- Step 4: Parse the XML file and obtain the Document.
  - The parse() method does it all in one call.

```
Document document = builder.parse( xmlFileURL );
```

© Copyright IBM Corporation 2004

Setting DOM Options

- Option settings are made to the DocumentBuilderFactory class prior to generating an implementing parser.

DOM Option Methods	Description
setNamespaceAware(Boolean) isNamespaceAware( )	Specifies that the parser will be Namespace aware. Default=false
setValidating(Boolean) isValidating( )	Specifies that the parser will validate the XML while parsing. Default=false
setCoalescing(Boolean) isCoalescing( )	Specifies if parser will convert CDATA nodes to Text nodes. Default=false
setExpandEntityReferences(Boolean) isExpandEntityReferences( )	Specifies if the parser will expand entity references. Default=true
setIgnoringComments(Boolean) isIgnoringComments( )	Specifies if the parser will ignore comments. Default=false
setIgnoringElementContentWhitespace(Boolean) isIgnoringElementContentWhitespace( )	Specifies if the parser will ignore "Ignorable" whitespace when parser is set to validating. Default=false
setAttribute(String name, Boolean) getAttribute(String name)	Sets/gets a particular attribute in the underlying DOM parser.

© Copyright IBM Corporation 2004

Getting DOM Options: ShowDOMInfo (1 of 2)

- This is the "Parser Creation..." chart in context:

```
public class ShowDOMInfo {

    public static void main(String args[]) {
        try {
            // Step 1: Create a DOM factory
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            checkSettings(dbf);

            // Step 2: Set any desired parser options
            dbf.setValidating(true);
            dbf.setNamespaceAware(true);

            // Step 3: Create a parser instance from the configured factory
            DocumentBuilder db = dbf.newDocumentBuilder();
            checkSettings(db);

            // Step 4: Parse the XML source and obtain the Document node
            Document doc = db.parse(args[0]);

            showDTDInfo(doc);

        }
        catch (javax.xml.parsers.ParserConfigurationException pce) {
(1)         System.out.println(
                "The parser was not configured correctly - " + pce.getMessage());
            System.exit(1);
        }
(2)     catch (java.lang.IllegalArgumentException ae) {
            System.out.println("Please specify an XML source - " + ae.getMessage());
            System.exit(1);
        }
(3)     catch (java.io.IOException ioe) {
            System.out.println("Error reading XML document - " + ioe.getMessage());
            System.exit(1);
        }
(4)     catch (org.xml.sax.SAXException se) {
            System.out.println("Error parsing document - " + se.getMessage());
            System.exit(1);
        }

    }
```

© Copyright IBM Corporation 2004

Exception Handling during Parsing

- Many things can go wrong when parsing a document.
- Common exceptions that can be thrown are:
  - javax.xml.parsers.ParserConfigurationException
    - Thrown when set to a bad configuration
  - java.lang.IllegalArgumentException
    - Thrown when bad arguments passed
  - java.io.IOException
    - Thrown when XML source cannot be opened
  - org.xml.sax.SAXException
    - Thrown when XML contains errors
- A SAX style error handler can be supplied to the DocumentBuilder using the setErrorHandler() method

© Copyright IBM Corporation 2004

Getting DOM Options: ShowDOMInfo (2 of 2)

- Note the two specifications of the checkSettings method:

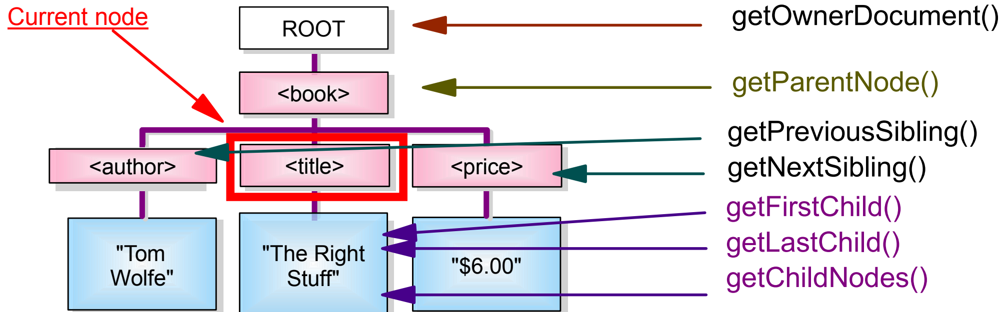
```
	private static void checkSettings(DocumentBuilderFactory
dbf) {
		System.out.println("Current DocumentBuilderFactory
Settings:");

		System.out.print("\tCoalescing: ");
		if (dbf.isCoalescing())
			System.out.println("True");
		else
			System.out.println("False");

		System.out.print("\tExpands Entity References: ");
		if (dbf.isExpandEntityReferences())
			System.out.println("True");
		else
			System.out.println("False");

		System.out.print("\tIgnoring Comments: ");
		if (dbf.isIgnoringComments())
			System.out.println("True");
		else
			System.out.println("False");

		System.out.print("\tIgnoring Whitespace: ");
		if (dbf.isIgnoringElementContentWhitespace())
			System.out.println("True");
		else
			System.out.println("False");

		System.out.print("\tNamespace Aware: ");
		if (dbf.isNamespaceAware())
			System.out.println("True");
		else
			System.out.println("False");

		System.out.print("\tValidating: ");
		if (dbf.isValidating())
			System.out.println("True");
		else
			System.out.println("False");
	}
```

```
private static void checkSettings(DocumentBuilder db) {
	System.out.println("Current DocumentBuilder Settings:");

	System.out.print("\tNamespace Aware: ");
	if (db.isNamespaceAware())
		System.out.println("True");
	else
		System.out.println("False");

	System.out.print("\tValidating: ");
	if (db.isValidating())
		System.out.println("True");
	else
		System.out.println("False");

	DOMImplementation domImpl = db.getDOMImplementation();

	System.out.print("\tStyle Sheets supported: ");
	if (domImpl.hasFeature("StyleSheets", "2.0"))
		System.out.println("True");
	else
		System.out.println("False");

}
```

© Copyright IBM Corporation 2004

Navigating the Document Tree

- Once a document is obtained from the parser, it is available for processing.
  - Begin at the root element node.
  - Recursively walk through the tree.
- Every node has getter methods to its relatives.
  - Document, Parent, Children, Siblings



© Copyright IBM Corporation 2004

DOM - Under the Covers (1 of 2)

- Most of the (DOM) APIs are **interfaces** rather than classes.
  - That means that
    - An implementation need only expose methods with the defined names and specified operation;
    - Not implement classes that correspond directly to the interfaces.
  - This allows the DOM APIs to be implemented
    - As a thin veneer on top of legacy applications with their own data structures, or
    - On top of newer applications with different class hierarchies.
- The consequence to us is ordinary constructors (in the Java or C++ sense) cannot be used to create DOM objects, since the underlying objects to be constructed may have little relationship to the DOM interfaces.
  - Conventional object-oriented design solution: define *factory methods* that create instances of objects that implement the various interfaces.
  - Objects implementing interface X are created by a createX() method on the Document interface; because all DOM objects live in the context of a specific Document.

© Copyright IBM Corporation 2004

DOM - Under the Covers (2 of 2)

- DOM does **not** address memory management issues, leaves these for the implementation.
  - Neither of the explicit language bindings defined by the DOM API (for ECMAScript and Java) require any memory management methods,
  - but DOM bindings for other languages (especially C or C++) may require such support.
- OMG IDL and ECMAScript have significant limitations in their ability to disambiguate names from different namespaces. . . so, DOM names tend to be long and descriptive in order to be unique across all environments.
- The DOM Core APIs present two different sets of interfaces to an XML/HTML document:
  - One presenting an object-oriented approach with a hierarchy of inheritance, and
  - A simplified view that allows all manipulation to be done via the Node interface without requiring (Java) casts.

© Copyright IBM Corporation 2004

Navigating DOM Sample: DOMUtil

- The next charts can be better understood in light of an executable example demonstrated using *Studio.*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: DOMUtil.java (executable)
- Input file: data/books.xml Output: -v

```
<!DOCTYPE books SYSTEM "books.dtd" >
<books>
    <book>
        <author>Tom Wolfe</author>
        <title>The Right Stuff</title>
        <price>$6.00</price>
    </book>
    <book>
        <author>R.L. Stevenson</author>
        <title>Treasure Island</title>
        <price>$13.00</price>
    </book>
    <book>
        <author>Carl Hiaasen</author>
        <title>Tourist Season</title>
        <price>$5.99</price>
    </book>
    <book>
        <author>Dave Barry</author>
        <title>Big Trouble</title>
        <price>$3.95</price>
    </book>
    <!-- Plus many, many more books -->
</books>
```

```
Console [<terminated> C:\j2sdk1.4.2_03\bin\javaw.exe (2/5/04 11:31 AM)]
DOCUMENT_NODE:
Name = #document
URI:LN  = null:null
Value = Null

+----Other Node Type:
+----Name = books
+----URI:LN  = null:null
+----Value = Null

+----ELEMENT_NODE:
+----Name = books
+----URI:LN  = null:books
+----Value = Null

+----+----TEXT_NODE:
+----+----Name = #text
+----+----URI:LN  = null:null
+----+----Value = (whitespace)

+----+----ELEMENT_NODE:
+----+----Name = book
+----+----URI:LN  = null:book
+----+----Value = Null

+----+----+----TEXT_NODE:
+----+----+----Name = #text
```

+ more...

© Copyright IBM Corporation 2004

Setting DOM Info: DOMUtil Code - Part I

- Compare the ShowDOMInfo.java chart with this DOMUtil.java chart:

Described on the next chart

```
public class DOMUtil {

    static int indent = 0;

    public static void main(String args[]) throws Exception {
        try {
            // Step 1: Create a DOM factory
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

            // Step 2: Set any desired parser options
            dbf.setValidating(true);
            dbf.setNamespaceAware(true);

            // Step 3: Create a parser instance from the configured factory
            DocumentBuilder db = dbf.newDocumentBuilder();

            // Step 4: Parse the XML source and obtain the Document node
            Document document = db.parse( args[0] );

            // Get the root element, even though its not used in this sample
            Element root = document.getDocumentElement();

            // Recurse through the document tree
            printTree(document);
        } catch (javax.xml.parsers.ParserConfigurationException pce) {
            System.out.println("The parser was not configured correctly - " + pce.getMessage() );
            System.exit(1);
(1)     } catch (java.lang.IllegalArgumentException ae) {
            System.out.println("Please specify an XML source - " + ae.getMessage() );
            System.exit(1);
(2)     } catch (java.io.IOException ioe) {
            System.out.println("Error reading XML document - " + ioe.getMessage() );
            System.exit(1);
(3)     } catch (org.xml.sax.SAXException se) {
            System.out.println("Error parsing document - " + se.getMessage() );
            System.exit(1);
(4)     }

    }
```

© Copyright IBM Corporation 2004

Traversing the DOM Tree: DOMUtil

- The first thing to do is obtain the document as a result of the parsing.

```
Document document = db.parse(args[0])
```

2. Next, get an instance of the root element.

```
Element root = document.getDocumentElement();
```

3. From there, it's just a matter of following the relationship references, and walking through the children branches.
   - Can be done by iterating through a list of nodes.

```
NodeList children = root.getChildNodes();
```

   - Or starting with the first child, and walking each sibling.

```
for( Node child = root.getFirstChild();
     child != null;
     child = child.getNextSibling() ) { ... }
```

© Copyright IBM Corporation 2004

Example: Recursing through the DOM Tree

- The most effective way to walk a DOM tree is through recursion as shown in this sample taken from *Studio*.

```
public static void printTree(Node node) {
    printNodeInfo( node );
    // Standard DOM recursion FOR loop
    for (Node child = node.getFirstChild();
        child != null;
        child = child.getNextSibling()) {
        indent++;           // Increment indent before next level
        printTree(child);
        indent--;           // Decrement indent after stepping up.
    }
}
```

Node org.w3c.dom.Node.getFirstChild()

The first child of this node. If there is no such node, this returns null.

- Calling this method and passing it the document node, will display the entire contents of the document.

```
...
Document document = domparser.parse( XMLFileURL );
printTree( document );
...
```

© Copyright IBM Corporation 2004

Getting DOM Info: DOMUtil - Part II

- This is the code for the printNodeInfo() method:

```
        public static void printNodeInfo(Node node) {

                String tab = "";
                // Add indentation to represent level
(1)             for (int i = 0; i < indent; i++)
                        tab += "+----";

(2)             switch (node.getNodeType()) {
                        case Node.DOCUMENT_NODE :
                                System.out.println( tab + "DOCUMENT_NODE:
");
                                break;
                        case Node.ELEMENT_NODE :
                                System.out.println( tab + "ELEMENT_NODE: ");
                                break;
                        case Node.TEXT_NODE :
                                System.out.println( tab + "TEXT_NODE: ");
                                break;
                        case Node.ATTRIBUTE_NODE :
                                System.out.println( tab + "ATTRIBUTE_NODE:
");
                                break;
                        case Node.COMMENT_NODE :
                                System.out.println( tab + "COMMENT_NODE: ");
                                break;
                        default :
                                System.out.println( tab + "Other Node Type: ");
                                break;
                }
```

Part 1 of 2

```
                System.out.println( tab + "Name = " + node.getNodeName());     (3)

                System.out.println( tab
                        + "URI:LN  = " + node.getNamespaceURI() + ":"
                        + node.getLocalName());

                String value = node.getNodeValue();                            (4)
                if (value == null)
                        System.out.println( tab + "Value = Null");
                else if (value.trim().length() == 0)
                        System.out.println( tab + "Value = (whitespace)");
                else
                        System.out.println( tab + "Value = " + value );

                // Is this an Element? If so, get any Attributes               (5)
                if ( node.getNodeType() == Node.ELEMENT_NODE ) {
                        NamedNodeMap attrs = node.getAttributes();
                        for( int a = 0; a < attrs.getLength(); a++ ) {         (6)
                                Attr attr = (Attr) attrs.item( a );
                                System.out.println( tab + "Attribute: " +
attr.getName()
                                        + " = " + attr.getValue() );
                        }

                }
                System.out.println( "" );
        }
```

Part 2 of 2

© Copyright IBM Corporation 2004

Working with Nodes: DOMUtil

- Once you have a node, what are you going to do with it?
  - Get basic information.

```
String name = child.getNodeName();
String value = child.getNodeValue();
```

  - Test the node to determine what type it is.
  - There are 12 basic node types.
    - The most common ones are shown below:

```
switch ( node.getNodeType() ) {
  case Node.DOCUMENT_NODE :
    System.out.println( "DOCUMENT_NODE: " + name );  break;
  case Node.ELEMENT_NODE :
    System.out.println( "ELEMENT_NODE: " + name ); break;
  case Node.TEXT_NODE :
    System.out.println( "TEXT_NODE: " + name );    break;
  case Node.ATTRIBUTE_NODE :
    System.out.println( "ATTRIBUTE_NODE: " + name ); break;
  case Node.COMMENT_NODE :
    System.out.println( "COMMENT_NODE: " + name );   break;
  default :
    System.out.println( "Other Node Type: " + name); break;
}
```

© Copyright IBM Corporation 2004

Dealing with Element Attributes: DOMUtil

- Attributes are treated as a special type of node.
- They are included within an element node.
- Can get all attributes as a NamedNodeMap collection
  - Retrieved using Node.getAttribues()
  - Each attribute can be extracted into an Attr Node
  - Attr is extended from Node Interface
- Get a single Attribute by name from an element node.
  - Get Attr: Element.getAttributeNode("AttributeName")
  - Get Value: Element.getAttribute("AttributeName")

```
// If this is an Element, display any attributes
if ( node.getNodeType() == Node.ELEMENT_NODE ) {
  NamedNodeMap attrs = node.getAttributes();
  for( int a = 0; a < attrs.getLength(); a++ ) {
    Attr attr = (Attr) attrs.item( a );
    System.out.println( tab + "Attribute: "
      + attr.getName() + " = " + attr.getValue() );
  }
}
```

© Copyright IBM Corporation 2004

Important DOM Odds and Ends: a Potpourri

- There are too many possible situations that may arise for us to cover every one of them
  - In spite of the depth of coverage so far, this *is* still an overview course.
- However, you will find many opportunities to use these:

Topic	**Class** (in dom.lecture)
Modifying node values	DiscountBooks.java
Adding nodes	AddBookSupplier.java
Creating a new DOM tree	AddNewBooks.java
Merging documents	MergeBooks.java

© Copyright IBM Corporation 2004

Modifying Node Values: DiscountBooks

- The next charts can be better understood in light of an executable example demonstrated using *Studio.*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: DiscountBooks.java (executable)
- Input [file:] data/salebooks.xml [value:] 25 [%] Output: -v

salebooks.xml

```xml
<books>
    <book>
        <author>Tom Wolfe</author>
        <title>The Right Stuff</title>
        <price onSale="No">6.00</price>
    </book>
    <book>
        <author>R.L. Stevenson</author>
        <title>Treasure Island</title>
        <price onSale="No">13.00</price>
    </book>
    <book>
        <author>Carl Hiaasen</author>
        <title>Tourist Season</title>
        <price onSale="No">5.99</price>
    </book>
    <book>
        <author>Dave Barry</author>
        <title>Big Trouble</title>
        <price onSale="No">3.95</price>
    </book>
    <!-- Plus many, many more books -->
</books>
```

Console [<terminated> C:\j2sdk1.4.2_03\bin\javaw.exe (2/6/

```
On Sale: $4.50  (was: $6.00)
On Sale: $9.75  (was: $13.00)
On Sale: $4.49  (was: $5.99)
On Sale: $2.96  (was: $3.95)
```

© Copyright IBM Corporation 2004

Modifying Node Values: DiscountBooks

- The value of a node can be changed using the Node.setNodeValue() method.

```
private static void markDown( Element root, double percent ) {
  // Get all "price" elements
  NodeList prices = root.getElementsByTagName("price");
  for( int i = 0; i < prices.getLength(); i++ ) {
    Element price = (Element) prices.item(i);
    // Only discount book if not already on sale!
    String onSale = price.getAttribute("onSale");
    if( "No".equals( onSale ) ) {
      // Get current value and discount it
      String oldPrice = price.getFirstChild().getNodeValue();
      String newPrice = calcNewPrice(oldPrice,percent );
      // Set node on sale and adjust new price
      price.setAttribute("onSale", "Yes");
      price.getFirstChild().setNodeValue( newPrice );
      System.out.println( "On Sale: $" + newPrice
        + "  (was: $" + oldPrice + ")");
    }
  }
}
```

© Copyright IBM Corporation 2004

Modifying Node Values: DiscountBooks (1 of 2)

```
        package dom.lecture;
(1)     import java.text.NumberFormat;
        import javax.xml.parsers.DocumentBuilder;
        import javax.xml.parsers.DocumentBuilderFactory;

(2)     import org.w3c.dom.Attr;
        import org.w3c.dom.Document;
        import org.w3c.dom.Element;
        import org.w3c.dom.NamedNodeMap;
        import org.w3c.dom.Node;
        import org.w3c.dom.NodeList;

        public class DiscountBooks {

(3)         static int indent = 0;

            public static void main(String args[]) throws Exception {

(4)             if (args.length < 2) {
                        System.out.println("Usage: salebooks.xml discount");
                        return;
                }
(5)             double percent = Double.parseDouble(args[1]);

(6)             // Step 1: Create and configure parser
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();

                // Step 2: Parse XML and retrieve Document
                Document document = db.parse(args[0]);
                Element root = document.getDocumentElement();

                // STEP 3:  Find price elements and put them on sale
                markDown(root, percent);
            }
```

Oops! Remnant from DOMUtil.java

© Copyright IBM Corporation 2004

Modifying Node Values: DiscountBooks (2 of 2)

```
          private static void markDown(Element rootNode, double percent) {

                  // Get all "price" elements
(7)               NodeList prices = rootNode.getElementsByTagName("price");
                  for (int i = 0; i < prices.getLength(); i++) {
                          Element price = (Element) prices.item(i);
                          // Only discount book if not already on sale!
(8)                       String onSale = price.getAttribute("onSale");
                          if ("No".equals(onSale)) {

                                  // Get current value and discount it
                                  String oldPrice =
(9)                                       price.getFirstChild().getNodeValue();
                                  String newPrice = calcNewPrice(oldPrice, percent);

                                  // Set node on sale and adjust new price
                                  price.setAttribute("onSale", "Yes");
(10)                              price.getFirstChild().setNodeValue(newPrice);

                                  System.out.println(
                                          "On Sale: $" + newPrice + "  (was: $" +
                                                  oldPrice + ")");
                          }
                  }
          }
```

(11)

```
private static String calcNewPrice(String oldPrice,
 double percent) {
        double price = Double.parseDouble(oldPrice);
        price = price - (price * (percent / 100));
        NumberFormat nf = NumberFormat.getInstance();
        nf.setMaximumFractionDigits(2);
        nf.setMinimumFractionDigits(2);
        return nf.format(price);
}
```

© Copyright IBM Corporation 2004

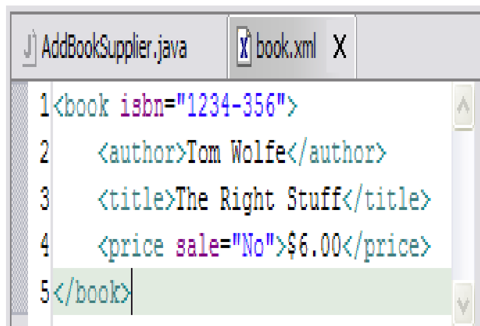
Modifying Node Values: DiscountBooks: Key

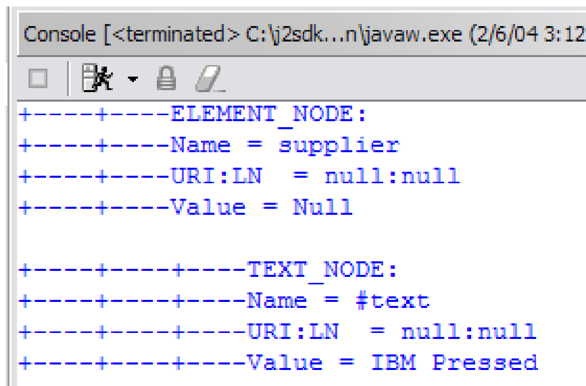
- (1) and (2) Support the code that follows.
  - You may recall NumberFormat from our introductory Java course.
  - It is used in (11) in the calcNewPrice() method.
- (3) Demonstrates a pitfall in code reuse.
- (4) Unlike earlier DOMUtil.java, we have very limited error checking.
- (5) Obtains the percentage by which we wish to reduce our oldPrice.
- (6) Is the block where we create and configure our document builder and parser tools. Compare the abbreviated approach we used here to those used in DOMUtil.java and ShowDOMInfo.java
- (7) and (8) Demonstrate the direct access methods described elsewhere.
- (9) Demonstrates an alternate method to obtain a value.
- (10) Demonstrates the actual modification of a node and an attribute.
- (11) Is the method that constructs our output.

© Copyright IBM Corporation 2004

Modifying Node Values: AddBookSupplier

- The next charts can be better understood in light of an executable example demonstrated using *Studio.*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: AddBookSupplier.java (executable)
- Input: data/book.xml Console Output: [produced 73 lines; last 9:]





© Copyright IBM Corporation 2004

Adding Nodes: AddBookSupplier

- Nodes can be added anywhere in the DOM tree.
  - Except element nodes to the document node
- Consists of creating a new branch of nodes and then grafting it onto the tree.
  - Start from the bottom and work up the branch
- Use a NodeList when a known set of elements are to be updated.

```
// Create a new Supplier node (text first, then element)
Text publisherText = document.createTextNode("IBM Pressed");
Node publisherNode = document.createElement("publisher");
// Add the text node to the element node
publisherNode.appendChild( publisherText );
// Create a list of books, and append an element to it
NodeList books = document.getElementsByTagName("book");
for( int i = 0; i < books.getLength(); i++ ) {
  Element book = (Element) books.item(i);
  book.appendChild( publisherNode );
}
```

© Copyright IBM Corporation 2004

Deleting and Replacing Nodes

- Nodes can be deleted from the tree using a single call.
  - Node.removeChild(Node nodeToBeRemoved)

```
// If book is out of print (publisher = none), remove supplier
if( node.getNodeName().equals("publisher") ) {
  String publisher = node.getFirstChild().getNodeValue();
  if( publisher == null || publisher.equals( "none" ) ) {
    node.getParentNode().removeChild( node );
  }
}
```

- Replacing one node with another is performed using:
  - Node.replaceNode(oldNode, newNode)

```
// Replace publisher w/supplier when a book goes out of print
Element supplier = document.createElement( "supplier" );
supplier.appendChild( document.createTextNode("Rare books") );
if( node.getNodeName().equals("publisher") ) {
  String publisher = node.getFirstChild().getNodeValue();
  if( publisher == null || publisher.equals("none") ) {
    node.getParentNode().replaceChild( node, supplier );
  }
}
```

© Copyright IBM Corporation 2004

Adding and Deleting Attributes

- Attributes can be added or replaced to an element.
  - By strings: Element.setAttribute(String name, String value)
  - By node: Element.setAttributeNode(Attr newAttrNode)

```
// Add "isbn" attribute to book elements, if not already there
private void addISBNAttribute( Element element ) {
  if( element.getNodeName().equals("book") ) {
    if ( ! element.hasAttribute("isbn") )
      element.setAttribute("isbn", "unset" );
  }
}
```

- Deleting an attribute can be done by name or node.
  - By name: Element.removeAttribute(String badAttr)
  - By node: Element.removeAttribute(Attr badAttrNode)

```
// Delete attribute by name in any element
private void zapAttr( Element element, String attr2zap ) {
  if ( element.hasAttribute( attr2zap ) )
    element.removeAttribute( attr2zap );
}
```

© Copyright IBM Corporation 2004

Creating a New DOM Tree: AddNewBooks

- The next charts can be better understood in light of an executable example demonstrated using *Studio.*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: AddNewBooks.java* (executable)
- Input: *no input* Console Output: [produced 81 lines]

*AddNewBooks.java extends DOMUtil.java

© Copyright IBM Corporation 2004

Creating a New DOM Tree (1 of 3): AddNewBooks

- Making a new DOM tree consists of:
  - Defining a new document
  - Creating a new root element
  - Creating branch of nodes (built from the bottom up)
  - Adding branches to the root
  - Adding root node to the document
- Step 1: Defining a new document:
  - Created from DocumentBuilder.newDocument()
  - Can reuse existing DocumentBuilder many times
- Step 2: Create a new root element:
  - Document.createElement("rootElementName");

```
DocumentBuilderFactory dbf =
  DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
// Step 1: Create new Document from DocumentBuilder
Document doc = db.doc.newDocument();
// Step 2: Create new "root" element from the Document
Element root = doc.createElement("books");
```

© Copyright IBM Corporation 2004

Creating a New DOM Tree (2 of 3): AddNewBook

- Step 3: Creating new branch of nodes:
  - Create new nodes from bottom up.
  - Attach each node to its parent node.

```
private static Node createNewBook( Document doc,
   String author, String title, String price )
{
  // Build new text and parent, add text to parent
  Text authorTxt = doc.createTextNode( author );
  Element authorNode = doc.createElement("author");
  authorNode.appendChild( authorTxt );
  // Shorten things a little
  Element titleNode = doc.createElement("title");
  titleNode.appendChild( doc.createTextNode( title ) );
  Element priceNode = doc.createElement("price");
  priceNode.appendChild( doc.createTextNode( price ) );
  // Now create the parent element and add all its children
  Element book = doc.createElement("book");
  book.appendChild( authorNode );
  book.appendChild( titleNode );
  book.appendChild( priceNode );
  return book;
}
```

© Copyright IBM Corporation 2004

Creating a New DOM Tree (3 of 3): AddNewBook

- Step 4: Add completed branch to root node:
  - "root" node is just another node.
- Step 5: Add completed root node to the document:
  - Final step in creating a document.

```
private static void main( String[] args ) {
   ...
   // Create new branch node and add to root
   Node book = createNewBook( doc,
     "Jimmy Buffett", "Tales from Margaritaville", "7.49" );
   root.appendChild( book );

   // Do it again, but shorten the process
   root.appendChild(
     createNewBook(
       doc, "John Steinbeck", "Cannery Row", "24.64")
   );

   // Finally, add completed root to document
   doc.appendChild( root );
   ...
}
```

© Copyright IBM Corporation 2004

Merging Documents: MergeBooks

- The next charts can be better understood in light of an executable example demonstrated using *Studio.*
- The source code for this file can be found in Package Explorer:
  - Project: XML Programming
  - Package: dom.lecture
  - Class: MergeBooks.java (executable)
- Input [2]: data/books.xml data/newbooks.xml
- Console Output: 381 lines

© Copyright IBM Corporation 2004

Merging Documents

- Parse or create two or more XML documents.
- Import node branch from source document into the target document.
  - Informs target document about node types.
- Insert imported node branch into appropriate place in target document tree.
- No way to validate changes in DOM level 2.

```
Document docAll = db.parse( args[0] );
Document docNew = db.parse( args[1] );
Element rootAll = docAll.getDocumentElement();
Element newRoot = docNew.getDocumentElement();
NodeList newBooks = newRoot.getElementsByTagNameNS(
  "http://www.ibm.com/ils/newbooks.html", "book" );
for( int i = 0; i < newBooks.getLength(); i++ ) {
  Element newBook = (Element) newBooks.item( i );
  Node newImportedBook = docAll.importNode( newBook, true );
  allRoot.appendChild( newImportedBook );
}
```

© Copyright IBM Corporation 2004

Writing DOM to Output

- DOM Level 2 specification has no defined methods for writing DOM output.
  - Left up to implementors.
  - Many issues with platform and language dependencies.
- Xerces provides sample program to output DOM.
  - DOMWriter.java
  - Completely manual process
- Use Document.normalize() to clean up a document.
  - Compresses consecutive text nodes into a single node.

© Copyright IBM Corporation 2004

Best Practices in DOM Programming (1 of 2)

- Program to the DOM API, not the implementation
  - Implementation methods usually append "Impl" to the Interface name
  - example: Use Document, not DocumentImpl
- Always use JAXP interfaces to abstract away the real parser implementation.
- Recursion is typically the best approach to traverse an entire DOM tree.
- Care should be taken when using entities to import data.
  - Could make the DOM grow exponentially.
- Keep a clean separation of concerns between the application's business logic and the actual parsing of the XML document.
- Write to the XML's vocabulary, not the data.

© Copyright IBM Corporation 2004

Best Practices in DOM Programming (2 of 2)

- Let the parser deal with validation, the structure and ordering of elements.
  - Don't duplicate the effort in your application.
  - Care should be taken to comply with the vocabulary when altering document structure.
    - DOM Level 2 has no way to validate after the parse.
- When using DTDs you have to perform application level validation of data.
  - Numbers may not be numbers
  - Dates may not be dates
  - Schema helps a lot here
- Use proper URIs on the DocumentBuilder.parse() method. Do not pass DOS file names
  - Good: file:///C:/xml/data/books.xml
  - Bad: C:\xml\data\book.xml

© Copyright IBM Corporation 2004

Checkpoint Questions (1 of 2)

1. DOM considers everything in an XML document as a:
   A. Element
   B. Tree Branch
   C. Node
   D. Event

2. Which of the following is not a valid DOM Node Type?
   A. Element Node
   B. Text Node
   C. Entity Node
   D. Root Node
   E. CDATASection Node

© Copyright IBM Corporation 2004

Checkpoint Questions (2 of 2)

3. Typically, the best way to traverse a DOM tree is:
   A. Recursion
   B. From the bottom up
   C. Event notification
   D. Using Document.getNextElement()

4. The DocumentBuilder can be used for (Select all that apply):
   A. DOM factory customization
   B. Parsing an XML document
   C. Populating a DOM tree from a Javabean
   D. Creating a new Document
   E. None of the above

© Copyright IBM Corporation 2004

Unit Summary

- DOM is a good choice when:
  - Modifying XML content
  - Changing document structure
  - Repeated use of the XML data
- DOM API represents an in-memory tree of the XML data and structure:
  - Provides methods for traversal, relationships, and modifications of content.
  - DOM API is complex but offers a lot of functionality.
- The JAXP API abstracts the actual implementation of the DOM API.
  - Apache Xerces parser provides full DOM implementation.

© Copyright IBM Corporation 2004