**Chapter 3:**

**Parsing and Creating XML Documents with DOM**

**What Is DOM?**

Document Object Model (DOM) is a Java API for parsing an XML document  
into an in-memory tree of nodes, and for creating an XML document from a  
node tree. After a DOM parser creates a tree, an application uses the DOM  
API to navigate over and extract info set items from the tree’s nodes.

- DOM is JAVA-Api for parsing XMLdoc to Nodes.

- DOM has two big advantages over SAX:

+ DOM permits random access to a document’s infoset items

+ DOM lets you create XML documents.

However, SAX is advantageous over DOM in that it can parse documents  
of arbitrary sizes, whereas the size of documents parsed or created by DOM  
is limited by the amount of available memory for storing the document’s  
node-based tree structure.

**A Tree of Nodes**

- DOM views an XML document as a tree: has a single root node and nodes.

-DOM permits nodes to exist that are not part of the tree structure. For  
example, an element node’s attribute nodes are not regarded as child nodes of  
the element node. Also, nodes can be created but not inserted into the tree; they  
can also be removed from the tree.

- DOM classifies nodes into 12 types: *Attribute node, CDATA section node, Comment node, Document node, Document fragment node, Document type node, Element node, Entity node, Entity reference node, Notation node, Processing instruction node, Text node.*

- Nonroot nodes never exist in isolation

**Exploring the DOM API**

- Obtain a DOM parser/document builder by first instantiating DocumentBuilderFactory:

- After the factory has been configured, call its DocumentBuilder newDocumentBuilder() method to return a document builder:

- If a document builder cannot be returned, this method throws a *ParserConfigurationException.*

Parsing and Creating XML Documents

- DocumentBuilder provides several overloaded parse() methods for parsing an XML document into a node tree.

- DocumentBuilder also declares the abstract Document newDocument() method for creating a document tree.

- Document is the root of the document’s node tree. It also declares various “create” and other methods for creating a node tree

- Node declares several methods for navigating the node tree:

+ boolean hasChildNodes() returns true when a node has child nodes.

+ Node getFirstChild() returns the node’s first child.

+ Node getLastChild() returns the node’s last child.

- Node declares four methods for modifying the tree:

+ Node insertBefore (Node newChild, Node refChild) inserts newChild before the existing node specified by refChild and returns newChild.

+ Node removeChild (Node oldChild) removes the child node identified by oldChild from the tree and returns oldChild.

+ Node replaceChild (Node newChild, Node oldChild) replaces oldChild with newChild and returns oldChild.

+ Node appendChild (Node newChild) adds newChild to the end of the current node’s child nodes and returns newChild.

- Document declares three methods for locating one or more elements:

+ Element getElementById(String elementId) returns the element that has an id attribute (as in <img id=...>) matching the value specified by elementId.

+ NodeList getElementsByTagName(String tagname) returns a nodelist of a document’s elements (in document order) matching the specified tagName.

+ NodeList getElementsByTagNameNS(String namespaceURI,String localName) is equivalent to the second method except in adding to the nodelist only those elements matching localName and namespaceURI values. Pass "\*" to namespaceURI to match all namespaces; pass "\*" to localName to match all local names.