Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-nine)

9.5: XML Parsing with DOM

The Java DOM (document object model) API is a tree based API. It parses an XML document and stores it in an internal tree representation. The Java DOM API also consists of several classes and interfaces to organize and store the XML data once it has been parsed.

In DOM, everything is a Node. All other interfaces, such as Document, Element and Text inherit from it. A Node is an object that can be the root or a leaf in a document. In addition, they can have child nodes. A Node includes interface methods for navigation as well as methods to get its name, type, attributes and children. The table below lists commonly used Node interface methods:

Table: Commonly used Node interface methods

Method	Descriptions
NodeList getChildNodes( )	Returns a list of all the children of the node
boolean hasChildNodes( )	Returns a boolean on whether the node has children or not
short getNodeType( )	Returns the type of the node. Types include in DOCUMENT_NODE, ELEMENT_NODE, TEXT_NODE, ATTRIBUTE_NODE (the complete list is defined in documentation for the Node interface).
String getNodeName( )	Returns the tag name of the node
Node getParentNode( )	Returns the parent of the node
NamedNodeMap getAttributes( )	Returns a NamedNodeMap of attributes of the node. Only works when the node is an Element type.

The Document interface represents the entire XML document. It is the entry point to an XML document's data. Its getChildNodes( ) method returns a list of all of the nodes in the documents. You can then traverse through each node. Note that getChildNodes( ) returns a NodeList type, which is not part of the standard Java Collections Framework (same with NamedNodeMap). Please see the documentation for more details on both of these collection types. We will show how to use NodeList in the example below.

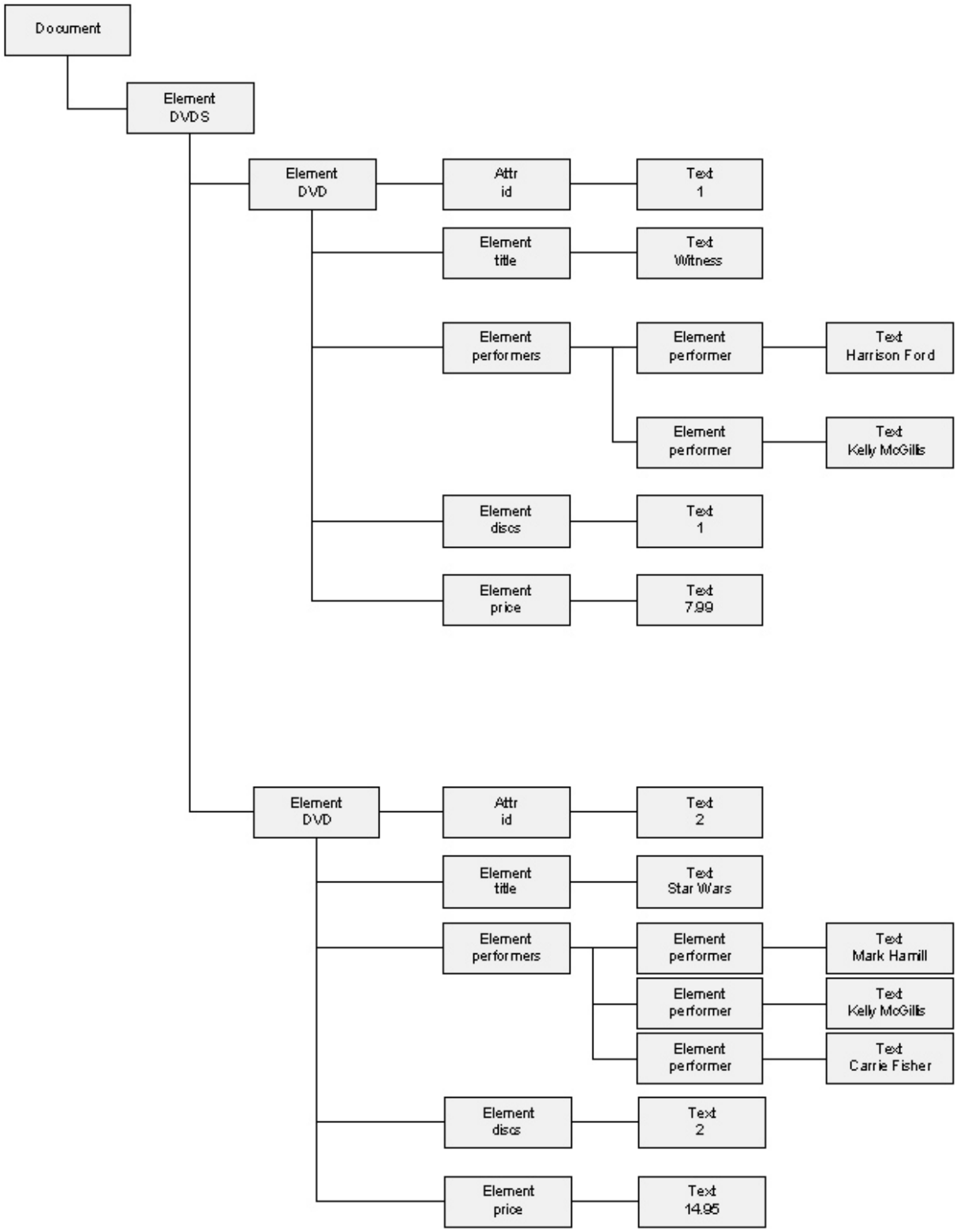
The Element interface represents an element in an XML document. You can get the tag name and attributes. See below for a list of its commonly used methods:

Table: Commonly used methods

Method	Description
String getTagName( )	Returns the name of the Element. (Same as getNodeName( ))
boolean hasAttribute(String name)	Returns a boolean on whether the attribute name exists
String getAttribute(String name)	Gets the attribute value by name

The value of an Element is not directly retrievable from the Element object. You would need to look for the Text node among its children first. The Text interface contains the text value within the element. You can get it using the getWholeText() method which returns the String value.

The diagram below shows our dvd.xml file represented as a DOM tree. It should help clarify the relationship between nodes in DOM.



The example below parses and displays the contents of dvd.xml using a DOM parser.

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import java.io.*;

public class DOMDVDReader {
  public void read(String filepath) {
```

```
  DocumentBuilderFactory dbf  = DocumentBuilderFactory.newInstance();
  dbf.setValidating(true);

  DocumentBuilder db = null;
  try {
     db =  dbf.newDocumentBuilder();
     db.setErrorHandler(new  ErrorHandler() {
      public void error(SAXParseException spe) {
                        System.err.println(spe);
                       }
      public void fatalError(SAXParseException spe){
                        System.err.println(spe);
                       }
      public void warning(SAXParseException spe) {
                        System.out.println(spe);
                       }
      });
     } catch (ParserConfigurationException  pce) {
               System.err.println(pce);
               System.exit(1);
     }

     Document doc = null;
     try {
       doc = db.parse(new File(filepath));
     }  catch (SAXException  se) {
       System.err.println(se);
     }  catch (IOException  ioe) {
       System.err.println(ioe);
     }
     NodeList nodeList =  doc.getDocumentElement().getChildNodes();
     EchoNodes(nodeList);
 }

 public void  EchoNodes(NodeList nodeList) {
 if (nodeList == null)  return;
 for (int i = 0; i <  nodeList.getLength(); i++) {
  Node child_node = nodeList.item(i);
     if  (child_node.getNodeType() == Node.ELEMENT_NODE) {
      Element el = (Element)child_node;
      System.out.println("Tag Name:" +  el.getTagName());
      NamedNodeMap attributes =  child_node.getAttributes();
      Node attribute = attributes.getNamedItem("id");
      if (attribute != null) System.out.println("Attr:  " + attribute.getNodeName() + " = " + attrib
ute.getNodeValue());
     } else if  (child_node.getNodeType() == Node.TEXT_NODE) {
      Text tn = (Text)child_node;
      String text = tn.getWholeText().trim();
      if (text.length() > 0) System.out.println("Text:  " + text);
     }
     EchoNodes(child_node.getChildNodes());
 }
 }

public static void  main(String[] args) {
 DOMDVDReader domDVDReader  = new DOMDVDReader();
```

```
 domDVDReader.read("dvd.xml");
 }
}
```

Let's go over the example step by step:

1. Import the DOM libraries:

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
```

2. In the read( ) method, we get a document builder factory instance.
3. We want it to validate the XML, so we invoke setValidating( ) to true.
4. Get a document builder object from the factory.
5. Set the error handlers.
6. We are now ready to parse the file. We pass the XML file as an argument to the document builder's parse method. This will return a Document object, assuming there were no errors.
7. Now that we have Document object, we can traverse through all the nodes. We pass the list of nodes using getChildNodes( ) to our EchoNodes( ) method.
8. EchoNodes( ) will recursively traverse through each node of the document and display its contents.

As you can see from the example, most code in a DOM parser is in the set up and error handling. Most of the work has already been done and you just need to call the appropriate methods to retrieve the data.

Note: The Java DOM API comes with several classes and interfaces that are not designed to fully take advantage of Java's features. This is because the designers of DOM did not want it to be tied down to any specific language. However, their usage is straightforward. Still, it is a good idea to have your documentation handy.