PARSING XML:

DOM BASED XML PROCESSING:

DOM:

- Include Java DOM API defined by org.w3c.dom package
- Nodes of DOM tree belong to classes such as Node, Document, Element, Text.
- Non-method properties of Node instances are accessed via methods

  parentNode is accessed by calling getParentNode()
- Methods such as getElementsByTagName() (that returned an arraylike list of Node instances in JavaScript) will in Java return an object implementing the NodeList interface.

PROGRAM:

DOM:

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import java.io.File;
public class Main {
   public static void main(String[] args)
   {
      try
      {
         File fXmlFile = new File("D:/Web Tech/Staff1.XML");
         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
         Document doc = dBuilder.parse(fXmlFile);
      System.out.println("RootSystem element :" + doc.getDocumentElement().getNodeName());
         NodeList nodes = doc.getElementsByTagName("staff");
         System.out.println("Input Elements has:" + nodes.getLength() + "nodes");
   }
      catch(Exception e)
```

```
            {
               System.out.println("XML Doc not Well formed" + e);
            }

         }
}
```

EVENT-ORIENTED PARSING: SAX-PARSER
SAX:

- An alternative approach is to have the parser interact with an application as it reads an XML document. This is the approach taken by SAX (Simple API for XML).
- SAX allows an application to register event listeners with the XML parser.
- SAX parser calls these listeners as events occur and passes them the information about the events.

Main.Java:

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
public class Main
{
    public static void main(String args[])
   {
   try
      {
      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser saxParser = factory.newSAXParser();
      saxParser.parse("D://Staff1.XML",new CountHelper());
      }
catch(Exception e)
   {
   e.printStackTrace();
   }
}
}
```

CountHelper.JAVA:

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class CountHelper extends DefaultHandler
{
int no_elms;
/*CountHelper()
{
super();
}*/
   @Override
public void startDocument() throws SAXException
   {
   no_elms=0;
   //return;
   }
   @Override
public void startElement(String u,String ln,String qname,Attributes atts)
          throws SAXException
   {
      if(qname.equals("firstname"))
      {
         no_elms++;
      }
     // return;
   }

 @Override
public void endDocument() throws SAXException
   {
```

```
        System.out.println("I/p Doc has " + no_elms + "firstname Elements");

    }
}
```

Staff1.xml:

```
<?xml version="1.0"?>
<company>
        <staff id="1001">
                <firstname>yong</firstname>
                <lastname>mook kim</lastname>
                <nickname>mkyong</nickname>
                <salary>100000</salary>
        </staff>
        <staff id="2001">
                <firstname>low</firstname>
                <lastname>yin fong</lastname>
                <nickname>fong fong</nickname>
                <salary>200000</salary>
        </staff>
</company>
```

OUTPUT:

DOM OUTPUT:

RootSystem element :company

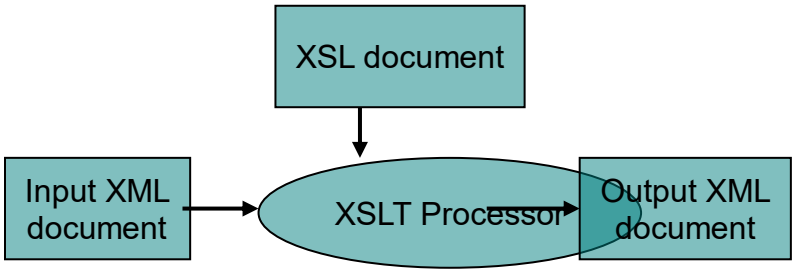
Input Elements has:2nodes

SAX OUTPUT:

I/p Doc has 2 firstname Elements

XSLT TRANSFORMATION: (XSLT)

- **The Extensible Stylesheet Language (XSL) is an XML vocabulary typically used to transform XML documents from one form to another form**



- JAXP allows a Java program to use the **Extensible Stylesheet Language (XSL)** to extract data from one XML document, process that data, and produce another XML document containing the processed data.

For example, XSL can be used to extract information from an XML document and embed it within an XHTML document so that the information can be viewed using a web browser.

TRANSFORMER:

Main.java:

```
import java.io.FileOutputStream;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
public class Main
{
   static String xml="D://WebTech//trans.XML";
   static String xslt="D://WebTech//trans.XSL";
   static String output="D://WebTech//trans.HTML";
   public static void main(String[] args)
   {
      try
      {
      TransformerFactory tf = TransformerFactory.newInstance();
      Transformer tr = tf.newTransformer(new StreamSource(xslt));
      tr.transform(new StreamSource(xml),new StreamResult(new FileOutputStream(output)));
```

```
        System.out.println("Output to " + output);
        }
        catch(Exception e)
        {
         }
    }
}
```

trans.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
         <body>
            <h1>Indian Languages details</h1>
            <table border="1">
               <tr>
                  <th>Language</th>
                  <th>Family/Origin</th>
                  <th>No. of speakers</th>
                  <th>Region</th>
               </tr>
       <xsl:for-each select="language">
               <tr>
                  <td><xsl:value-of select="name"/></td>
                  <td><xsl:value-of select="family"/></td>
                  <td><xsl:value-of select="users"/></td>
                  <td><xsl:value-of select="region"/></td>
               </tr>
             </xsl:for-each>
            </table>
         </body>
```

```
        </html>
    </xsl:template>
</xsl:stylesheet>
```

trans.xml:

```
<?xml version="1.0"?>
<!--<?xml-stylesheet type="text/xsl" href="trans.xsl"?>-->
<language>
<name>Kannada</name>
<region>Karnataka</region>
<users>38M</users>
<family>Dravidian</family>
</language>
```

trans.html:

Indian Languages details

Language	Family/Origin	No. of speakers	Region
Kannada	Dravidian	38M	Karnataka

XPATH:

XPath Nodes:

In XPath, there are seven kinds of nodes:

element, attribute, text, namespace, processing-instruction, comment, and document nodes.

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Relationship of Nodes:

- Parent
- Children
- Siblings
- Ancestors

- Descendants

XPath axes

An XPath axis is a path through the node tree making use of particular relationship between nodes. We use the "child::*" axis and the "attribute::*" axis all the time but mostly their short form: "*" and "@*".

1. Axis examples

Most often the axes, e.g.: "descendant::", are followed by an "*" (element) or an element name
but comment(), processing-instruction() andtext-node() can also be used. Only root node and elements can have children.

The construct node() only selects nodes being children of other nodes: element(), text(), comment(), processing-instruction(). An attribute has a parent element but attribute and namespace nodes are not children and not included in the node() construct or type.

descendant::*	All elements being children and children's children, etc, of context node.
descendant::stock	All "stock" elements being children and children's children, etc, of context node.
descendant::comment()	All comment nodes being children of context node.
descendant::*|comment()	All element and comment nodes being children or children's children, etc, of context node.
descendant::node()	All nodes being children or children's children, etc, of context node.

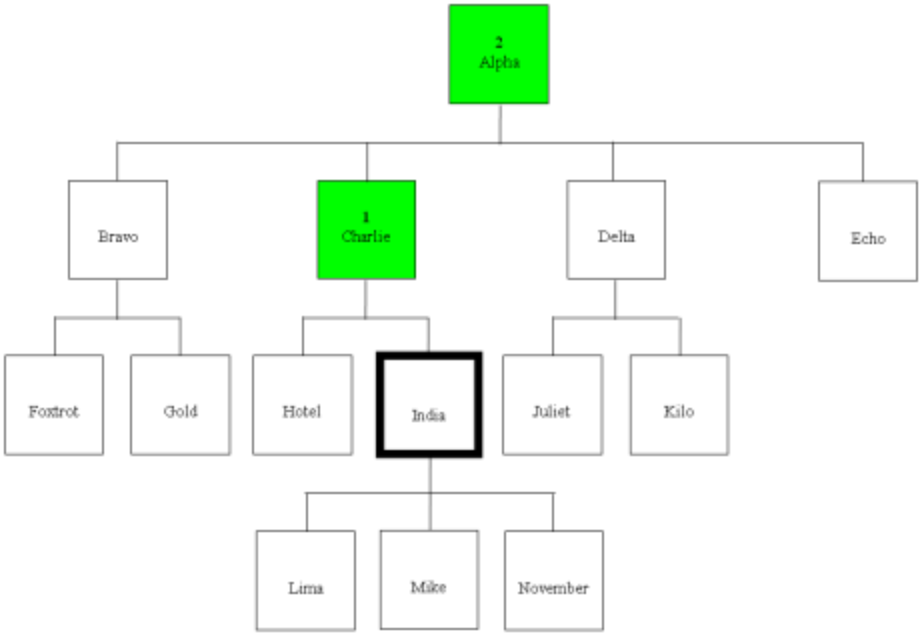
2. Axis diagrams

The diagrams are not showing attribute() and namespace() nodes and I have not made diagrams for the attribute and namespace axes. The namespace axis is deprecated. The functions in-scope-prefixes() and namespace-uri-for-prefix() should be used instead.

When using axes it is often important to remember that the nodes are processed in document order (indicated below). If both a node's children and siblings are included in an axis, the node's children are always processed before its following siblings. The order is sometimes a little tricky especially for reverse axes until you get used to it. See the preceding axis as example.

2.1 ancestor

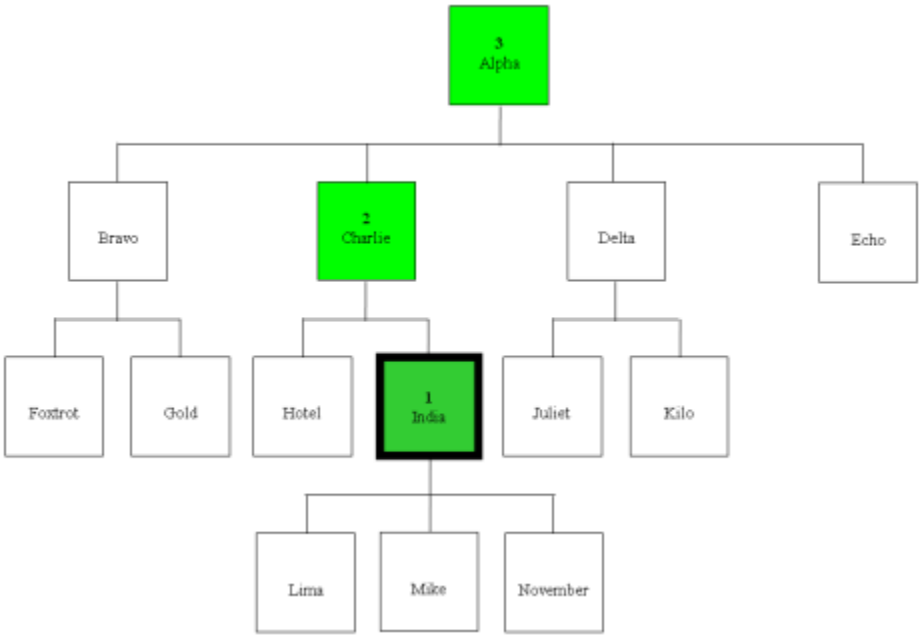
Ancestor axis: parent and parent's parent, etc.



India is context node: count(ancestor::*) returns 2.

2.2 ancestor-or-self

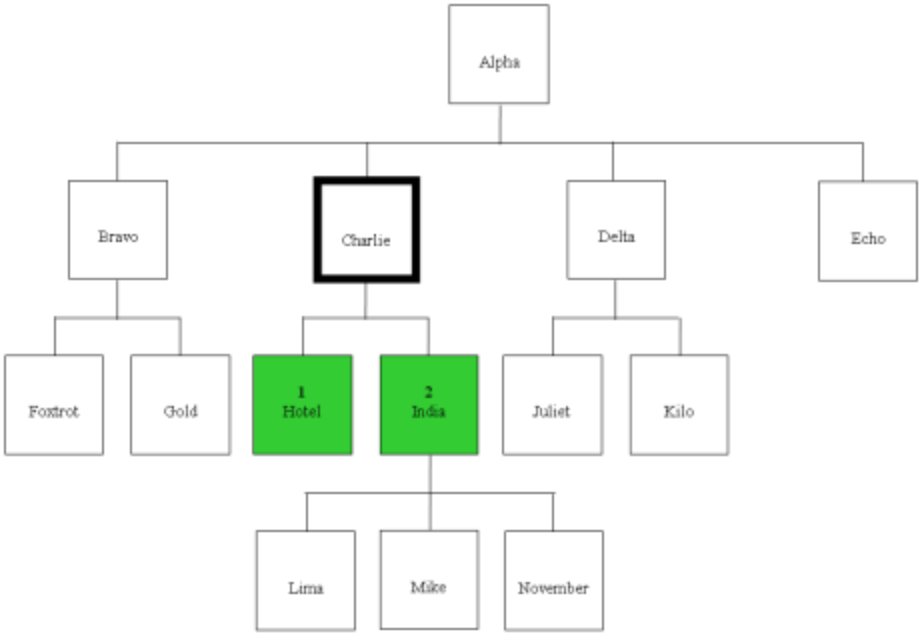
Ancestor-or-self axis: context node and parent and parent's parent, etc.



India is context node: count(ancestor-or-self::*) returns 3.

2.3 child

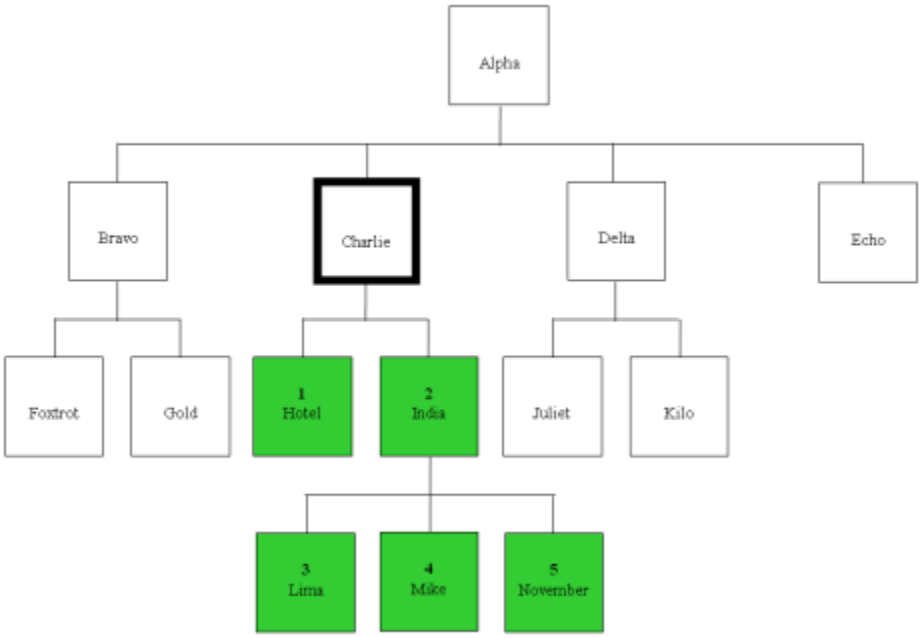
Child axis: children of context node.



Charlie is context node, e.g.: count(child::*) returns 2. Same as count(*).

2.4 descendant

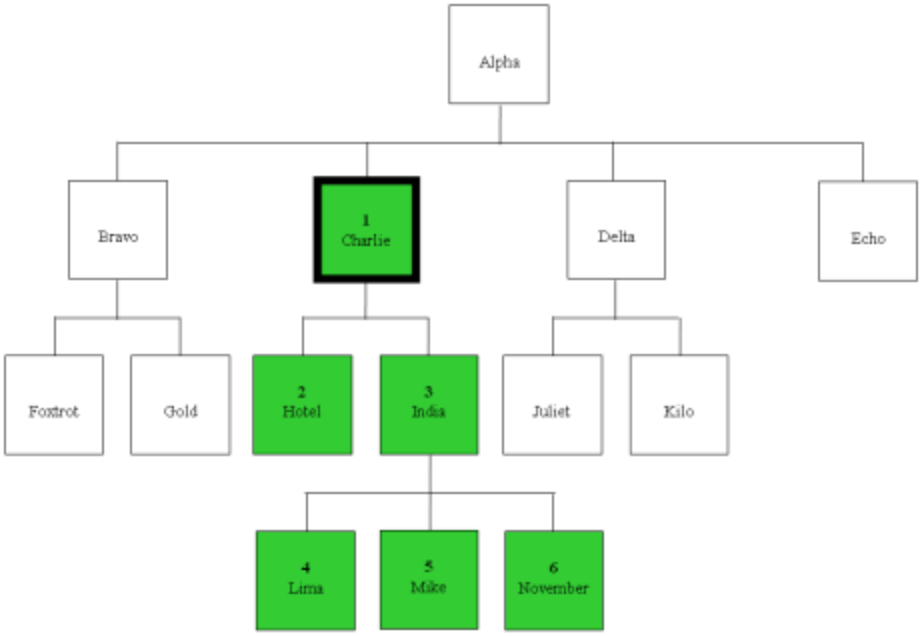
Descendant axis: children and their children, etc.



Charlie is context node, e.g.: count(descendant::*) returns 5.

2.5 descendant-or-self

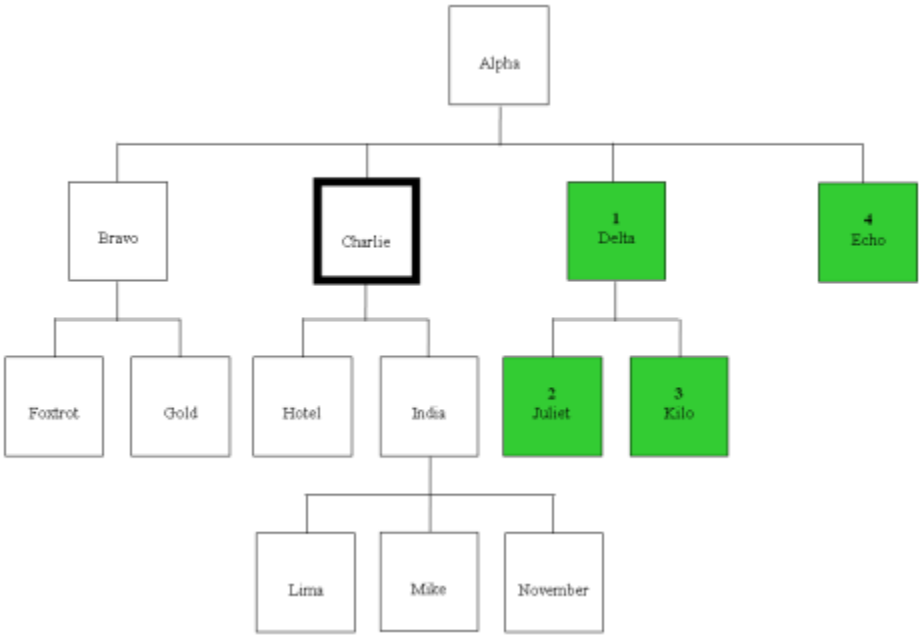
Descendant-or-self axis: context node and children and their children, etc.



Charlie is context node, e.g.: count(descendant-or-self::*) returns 6.

2.6 following

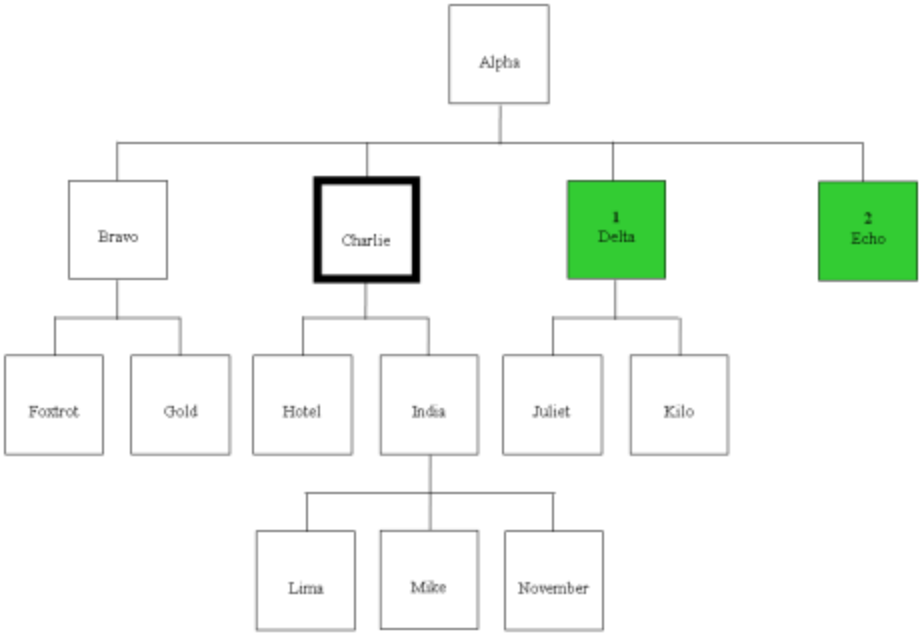
Following axis: following siblings and their children and their children, etc.



Charlie is context node, e.g.: count(following::*) returns 4.

2.7 following-sibling

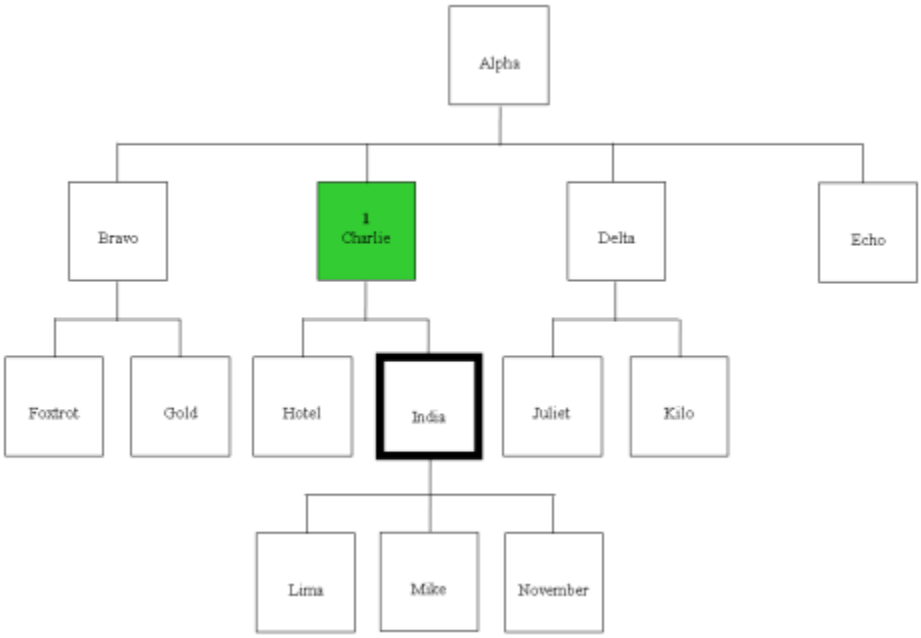
Following-sibling axis: following siblings.



Charlie is context node, e.g.: count(following::*) returns 2.

2.8 parent

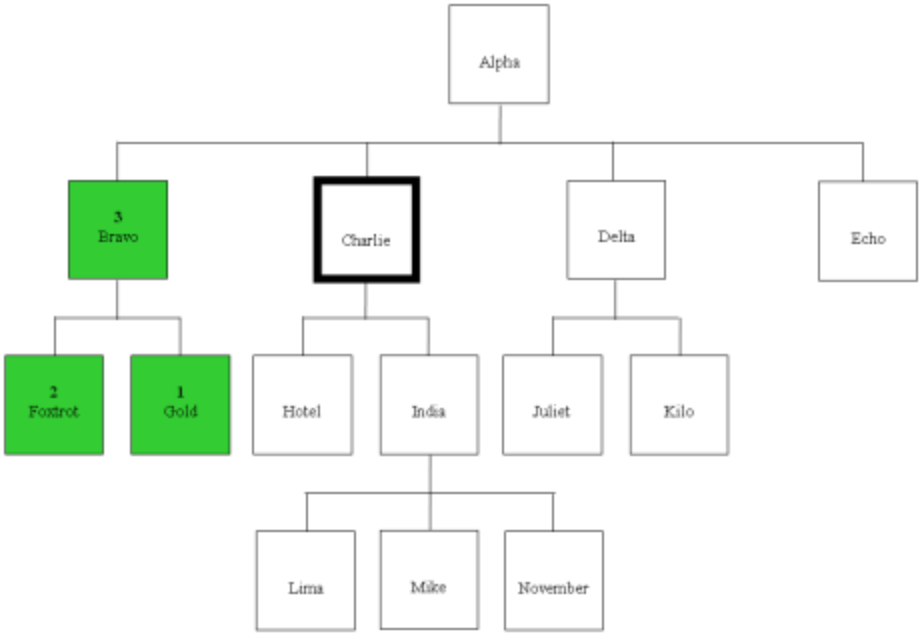
Parent axis: The parent of context node.



India is context node, e.g.: if (parent::* eq 'Charlie') then '...' else '...'

2.9 preceding

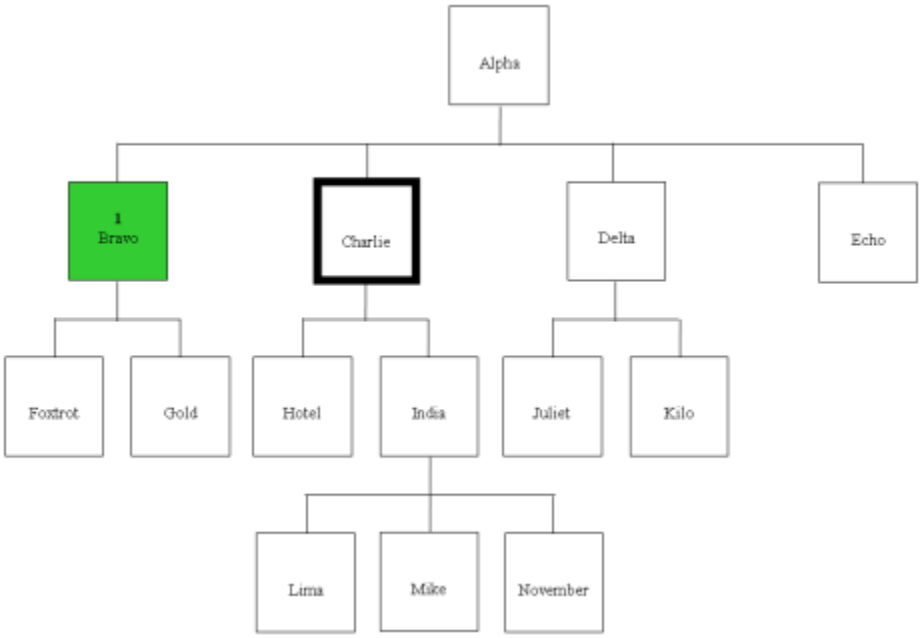
Preceding axis: preceding siblings and their children and their children, etc.



Charlie is context node, e.g.: count(preceding::*) returns 3

2.10 preceding-sibling

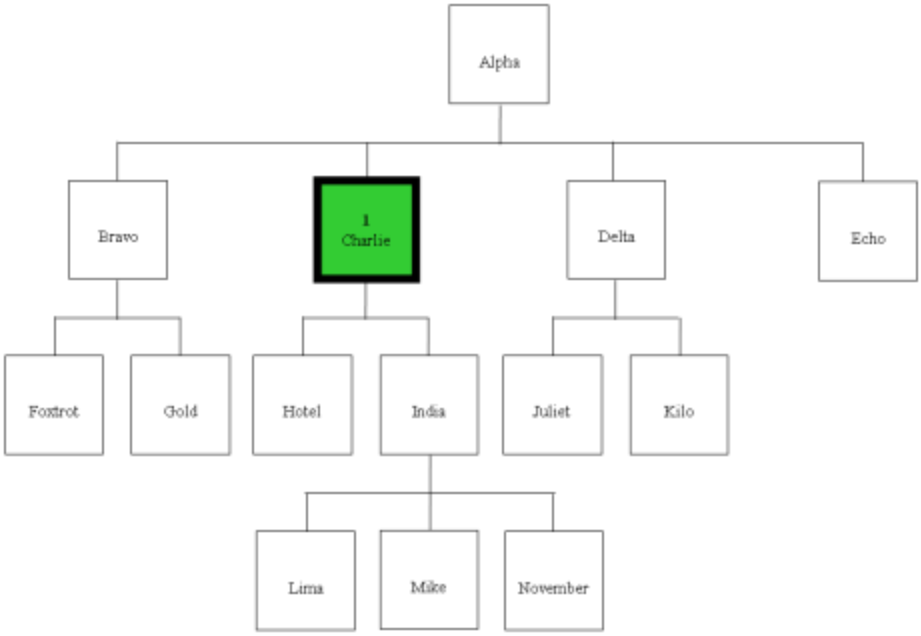
Preceding-sibling axis: preceding siblings.



Charlie is context node, e.g.: count(preceding-sibling::*) returns 1

2.11 self

Self axis: context node.



Charlie is context node, e.g.: if (self::* eq 'Charlie') then '...' else '...'

XML:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Edited by XMLSpy® -->

<catalog>

<cd>

<title>Empire Burlesque</title>

<artist>Bob Dylan</artist>

<country>USA</country>

<company>Columbia</company>

<price>10.90</price>

<year>1985</year>

</cd>

</catalog>

XSL:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Edited by XMLSpy® -->

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="count(descendant::node())" /></td>
        <td><xsl:value-of select="child::artist" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

OUTPUT:

Title	Artist
19	Bob Dylan

descendant::node()=19; Which includes all nodes like text,element,comment etc

descendant::text()=13; Which includes all text nodes alone.

<catalog>

<cd>**(t1)**

<title>Empire Burlesque**(t2)**</title>**(t3)**

<artist>Bob Dylan**(t4)**</artist>**(t5)**

<country>USA**(t6)**</country>**(t7)**

<company>Columbia**(t8)**</company>**(t9)**

<price>10.90**(t10)**</price>**(t11)**

<year>1985**(t12)**</year>**(t13)**

</cd>

</catalog>

Total **text** nodes:13

Total nodes: Text nodes**(13)**+Elements**(6)=19**

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
Nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter w they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

ABBREVIATIONS

Parent::node()=**..(two dots)**

Self::node()**=.(single dot)**

[attribute display=”visible”]=> [@display=”visible”]

THE <XSL:APPLY-TEMPLATES> ELEMENT

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the <xsl:apply-templates> element it will process **only the child element that matches the value of the attribute**. We can use the select attribute to specify the order in which the child nodes are processed.

It will not process the other nodes that are not in the select attribute list.

XML:

```
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
</catalog>
```

XSL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
    <xsl:apply-templates select="title"/>
```

```
    <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select=".."/></span>
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
</xsl:stylesheet>
```

Title: Empire Burlesque Bob Dylan USA Columbia 10.90 1985 **(Parent)**

Artist: Bob Dylan**(Self)**

Modeling Databases in XML:

Fundamental properties of XML that make it different from the relational model:

- **XML is self-describing.** A given document contains not only the data, but also the necessary metadata. As a result, an XML document can be searched or updated without requiring a static definition of the schema. Relational models, on the other hand, require more static schema definitions. All the rows of a table must have the same schema.
- **XML is hierarchical.** A given document represents not only base information, but also Information about the relationship of data items to each other in the form of the hierarchy. Relational models require all relationship information to be expressed either by primary key or foreign key relationships or by representing that information

in other relations.

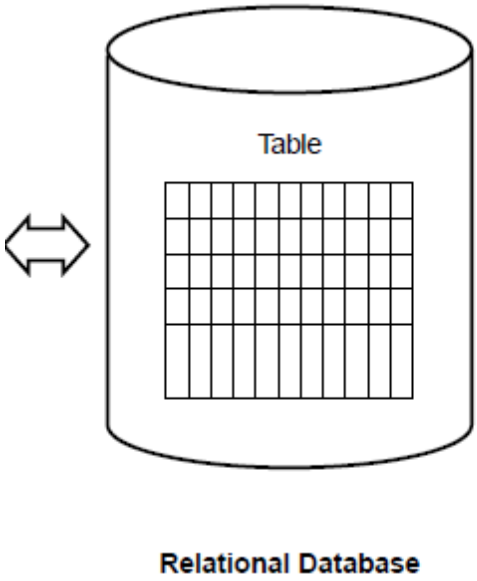
- **XML is sequence-oriented—order is important. Relational models are set oriented—order is unimportant.**

None of these differences indicate that XML is better or worse than purely relational models. In fact, XML and relational models are complementary solutions. Some data is inherently hierarchical while other data is inherently tabular; some data has more rigid schema while other data has less rigid schema; some data needs to follow a specific order while other data does not

XML Database Mapping

The first type of XML database solution provides a mapping between the XML document and the database fields. The system dynamically converts SQL result sets to XML documents. Depending on the sophistication of the product, it may provide a graphical tool to map the database fields to the desired XML elements. Other tools support a configuration file that defines the mapping. These tools continue to store the information in relational database management system (RDBMS) format. They simply provide an XML conversion process that is normally implemented as a server-side Web application.

```
<rental_property>
<prop_id>1</prop_id>
<name>The Meadows</name>
<address>
<street>251 Eisenhower Blvd</street>
<city>Houston</city>
<state>TX</state>
<postal_code>77033</postal_code></address>
<square_footage>500.0</square_footage>
<bedrooms>1.0</bedrooms>
<bath>1.0</bath>
<price>600</price>
<contact>
<phone>555-555-1212</phone>
<fax>555-555-1414</fax?
</contact>
</rental_property>
```



Relational Database

In the JAXB framework, we can parse XML documents into a suitable Java object. This technique is referred to as *unmarshaling*. The JAXB framework also provides the capability to generate XML documents from Java objects, which is referred to as *marshaling*.

Mapping xml with relational schema:

1. Review the database schema.
2. Construct the desired XML document.
3. Define a schema for the XML document.
4. Create the JAXB binding schema.
5. Generate the JAXB classes based on the schema.
6. Develop a Data Access Object (DAO).
7. Develop a servlet for HTTP access.