XML Introduction

• Evolution of markup languages:
  - GML → SGML → HTML
• eXtensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable
• XML was designed to describe data, with focus on what data is

Session objectives

Introduction
Parsing XML document
  - Using SAX
  - Using DOM
XML transformation

XML Features

• XML is a markup language much like HTML
• XML was designed to describe data
• XML tags are not predefined. You must define your own tags
• XML uses a Document Type Definition (DTD) or an XML Schema to describe the data
• XML with a DTD or XML Schema is designed to be self-descriptive
• A document consists of one outermost element, called root element that contains all the other elements, plus some optional administrative information at the top, known as XML declaration
Sample XML document

```xml
<student id="2464353">
  <fullname>Vo Van Hai</fullname>
  <address>
    <no>12</no>
    <street>Nguyen Van Bao</street>
    <district>Go Vap</district>
  </address>
  <email>vovanhaiQN@gmail.com</email>
</student>
```

Quick XML review

• Syntax
  o Every XML document has a preamble
    • `<?xml version="1.0" ?>`
  o An XML document may or may not have a DTD (Document Type Definition) or Schema
    • `<!DOCTYPE catalog>`
  o Every element has a start and end tag, with optional attributes
    • `<catalog version="1.0" />`
  o If an element does not contain any data (or elements) nested within, the closing tag can be merged with the start tag like so:
    • `<catalog version="1.0"/>`

• Syntax cont.
  o Elements must be properly nested
  o The outermost element is called the root element
  o An XML document that follows the basic syntax rules is called well-formed
  o XML documents do not always require a DTD or Schema, but they must be well-formed

Well-formed and valid XML document

• A "Well-formed" XML document has correct XML syntax
  o XML documents must have a root element
  o XML elements must have a closing tag
  o XML tags are case sensitive
  o XML elements must be properly nested
  o XML attribute values must be quoted
• An XML document that is well-formed and conforms to a DTD or Schema is called valid
Benefits of XML

- Data independence: separates the content from its presentation
- Easier to parse: frameworks for data exchange
- Reducing server load: using DOM to manipulate the data
- Easier to create: it is text-based
- Web site content: transforms to HTML using XSLT and CSS
- Remote procedure call: allows distributed computing
- E-commerce: sends data from one company to another

XML Namespace

- In XML, element names are defined by the developer
- This often results in a conflict when trying to mix XML documents from different XML applications
- XML Namespaces provide a method to avoid element name conflicts

XML Namespace example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore xmlns="http://www.w3.org/TR/htm4/">
    <table>
        <tr>
            <td>Apples</td>
            <td>Bananas</td>
        </tr>
    </table>
    <table>
        <tr><td>Coffee Table</td></tr>
        <td width="80"></td>
        <td width="120"></td>
    </table>
</bookstore>
```

Document Type Definition (DTD)

- The purpose of a DTD is to define the structure of an XML document
- A DTD document defines:
  - The legal building blocks of an XML document
  - The document structure with a list of legal elements and attributes
  - The way elements relate to one another within the document's tree structure
- A DTD can be declared:
  - Inline inside an XML document
  - External reference
Structure of a DTD

1. **DOCTYPE declaration**
   - Specifies name of that DTD and either its content or location

2. **ELEMENT declaration**
   - Specifies the name of the element, the content which that element can contain

3. **ATTRIBUTE declaration**
   - Specifies the element that owns the attributes, the attribute name, its type and its default values (if any)

4. **ENTITY declaration**
   - Specifies the name of the entity and either its value or location of its values.

Example DTD

```xml
<!DOCTYPE thuVien [  
  <!ELEMENT thuVien (sach+)>  
  <!ELEMENT sach (id, ten, tacGia, nhaXuatBan?, gia?)>  
  <!ELEMENT id (#PCDATA)>  
  <!ELEMENT ten (#PCDATA)>  
  <!ELEMENT tacGia (#PCDATA)>  
  <!ELEMENT nhaXuatBan (#PCDATA)>  
  <!ELEMENT gia (#PCDATA)>  
  <!ENTITY theLoai (Khoa hoc|Giao trinh|Tinh hoc)>  
  <!ENTITY nongNam CDATA IMPLIED>  
  <!ENTITY nxbt "Nhà xuất bản frê">  
  <!ENTITY nxbgd "Nhà xuất bản Giáo dục"> ]>
```

Example of internal DTD declaration

- If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition
- Syntax:  
  ```xml
  <!DOCTYPE root-element [element-declarations]>
  ```

Example of external DTD declaration

- If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition
- Syntax:  
  ```xml
  <!DOCTYPE root-element SYSTEM "filename"/>
  ```
XML Schema

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the order of child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

DTD limitations and XML Schema

- An objective of XML schema: overcome the drawbacks of DTDs
  - DTDs are written in a non-XML syntax
  - DTDs are not extensible
  - They have no support for namespaces
- The XML Schema language is referred as XML Schema Definition (XSD)

Example of XML Schema

```xml
<xml version="1.0">
  <Blog>
    <Title>Let me know what you think</Title>
    <Author>Yin Yang</Author>
  </Blog>
</xml>
```

Referencing a Schema in an XML Document

```xml
<xml version="1.0">
  <Blog xmlns="http://www.w3schools.com">
    <Title>Let me know what you think</Title>
    <Author>Yin Yang</Author>
  </Blog>
</xml>
```
JAXP - Java API for XML Processing

- JAXP is a collection of APIs that you can use in your Java applications to process and translate XML documents
- Consists of three APIs:
  - Simple API for XML (SAX): Allows you to use a SAX parser to process the XML documents serially
  - Document Object Model (DOM): Allows you to use a DOM parser to process the XML documents in an object-oriented manner
  - XML Stylesheet Language for Transformation (XSLT): Allows you to transform XML documents in other formats, such as HyperText Markup Language (HTML)
SAX

- Simple API for XML
- Started out as a Java API, but now exists for other languages too
- Quantity of memory usage is low
- Only for reading xml-documents
- Uses a event-driven model

Parsing an XML document

- The three steps followed for parsing an XML document are:
  - Application supplies a SAX content handler to the parser
  - Application tells parser to start parsing a document
  - Parser calls back the ContentHandler/DefaultHandler

Callbacks

- SAX works through callbacks: you call the parser, it calls methods that you supply

Your program

```
main(…)
```

The SAX parser

```
startDocument(…)
startElement(…)
characters(…)
endElement( )
endDocument( )
```

XML document

```
<library>
  <book>
    <title>
      Harry Potter
    </title>
    <price>
      35.0
    </price>
  </book>
  ....
</library>
```

Callback interface

- SAX is used with XMLReader as the Subject and org.xml.sax.ContentHandler/ org.xml.sax.helpers.DefaultHandler as an Observer (is similar to define the event Listener to subject – button in AWT)

```
startDocument()
startElement("library", attr)
startElement("book", attr)
startElement("title", attr)
characters(char[], start, length)
endElement("title")
... 
endElement("library")
endDocument()
```
Obtaining a SAX parser

- Important classes
  - `javax.xml.parsers.SAXParserFactory`;
  - `javax.xml.parsers.SAXParser`;
  - `javax.xml.parsers.ParserConfigurationException`;

- SAX parser
  - //gets a factory to make parsers
    ```java
    SAXParserFactory factory = SAXParserFactory.newInstance();
    ```
  - //gets a parser from the factory
    ```java
    SAXParser saxParser = factory.newSAXParser();
    ```
  - //parse the document
    ```java
    saxParser.parse( new File(argv[0]), handler);
    ```

---

ContentHandler interface (1)

- The `org.xml.sax.ContentHandler` interface:
  - Receives notifications of the logical content of the document being parsed
  - Application can implement this interface and register an instance with the SAX parser using the `setContentHandler()` method
  - The parser uses this instance to report events
  - The order of events in this interface is very important and it mirrors the order of information in the document itself
  - The different methods must be implemented

---

ContentHandler interface (2)

```java
import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXException;

public class MyContentHandler implements ContentHandler {
    public void setDocumentLocator(Locator locator) {}
    public void startDocument() throws SAXException {}
    public void endDocument() throws SAXException {}
    public void startPrefixMapping(String prefix, String uri) throws SAXException {}
    public void endPrefixMapping(String prefix) throws SAXException {}
    public void characters(char[] ch, int start, int length) throws SAXException {}
    public void ignorableWhitespace(char[] ch, int start, int length) {}
    public void processingInstruction(String target, String data) throws SAXException {}
    public void skippedEntity(String name) throws SAXException {}
}
```

---

DefaultHandler class (1)

- The `org.xml.sax.helpers.DefaultHandler` class:
  - Provides default implementations for all of the callbacks in the four core SAX2 handler classes: ContentHandler, DTD Handler, Error Handler, Entity Resolver
  - The application developer will need to override only those for which additional or custom functionality is desired
Overriding Handler methods

• Most important methods to override
  o void startDocument()  
    • Called once when document parsing begins
  o void endDocument()  
    • Called once when parsing ends
  o void startElement(...)  
    • Called each time an element begin tag is encountered
  o void endElement(...)  
    • Called each time an element end tag is encountered
  o void characters(...)  
    • Called randomly between startElement and endElement calls to
      accumulated character data

The Handler class (1)

public class Handler extends DefaultHandler
{
    public void startElement(String uri, String localName, String qName, Attributes attributes)
    throws SAXException {
        System.out.println("startElement: "+ qName);
    }
}

The Handler class (2)

// SAX calls this method to pass in character data
public void characters(char[] ch, int start, int length)  
    throws SAXException {
    System.out.println("characters: "+ new String(ch, start, length) + ";");
}
// SAX call this method when it encounters an end tag
public void endElement(String uri, String localName, String qName)  
    throws SAXException {
    System.out.println("Element: /" + qName);
}

Results

• If the file hello.xml contains:
  
  <display>Hello World!</display>

• Then the output from running java Sample will be:
  startElement: display
  characters: "Hello World!"
  Element: /display
More results

• Now suppose the file hello.xml contains:
  ```xml
  <?xml version="1.0"?>
  <display>
    <i>Hello</i> World!
  </display>
  ```

• Notice that the root element, <display>, now contains a nested element <i> and some whitespace (including newlines)

```
startElement: display
characters: // empty string
characters: // tab
startElement: i
characters: Hello
endElement: /i
characters: World!
endElement: /display
```

Simple SAX program

• The program consists of two classes:
  a. Sample -- This class contains the main method; it
    • Gets a factory to make parsers
    • Gets a parser from the factory
    • Creates a Handler object to handle callbacks from the parser
    • Tells the parser which handler to send its callbacks to
    • Reads and parses the input XML file
  b. Handler -- This class contains handlers for three kinds of callbacks:
    • startElement callbacks, generated when a start tag is seen
    • endElement callbacks, generated when an end tag is seen
    • characters callbacks, generated for the contents of an element

```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Sample {
  public static void main(String args[]) throws Exception {
    // Create a parser factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Make the parser
    SAXParser saxParser = factory.newSAXParser();
    XMLReader parser = saxParser.getXMLReader();
    // Create a handler
    Handler handler = new Handler();
    // Tell the parser to use this handler
    parser.setContentHandler(handler);
    // Finally, read and parse the document
    parser.parse("hello.xml");
  }
}
```

You will need to put the file hello.xml:
  a. In the same directory, if you run the program from the command line
  b. Or where it can be found by the particular IDE you are using

The Sample class (1)

```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Sample {
  public static void main(String args[]) throws Exception {
    // Create a parser factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Make the parser
    SAXParser saxParser = factory.newSAXParser();
    XMLReader parser = saxParser.getXMLReader();
```

The Sample class (2)

```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Sample {
  public static void main(String args[]) throws Exception {
    // Create a parser factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Make the parser
    SAXParser saxParser = factory.newSAXParser();
    XMLReader parser = saxParser.getXMLReader();
```

You will need to put the file hello.xml:
  a. In the same directory, if you run the program from the command line
  b. Or where it can be found by the particular IDE you are using

The Sample class (3)

```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Sample {
  public static void main(String args[]) throws Exception {
    // Create a parser factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Make the parser
    SAXParser saxParser = factory.newSAXParser();
    XMLReader parser = saxParser.getXMLReader();
    // Create a handler
    Handler handler = new Handler();
    // Tell the parser to use this handler
    parser.setContentHandler(handler);
    // Finally, read and parse the document
    parser.parse("hello.xml");
  }
}
```

You will need to put the file hello.xml:
  a. In the same directory, if you run the program from the command line
  b. Or where it can be found by the particular IDE you are using

The Sample class (4)

```
import javax.xml.parsers.*; // for both SAX and DOM
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Sample {
  public static void main(String args[]) throws Exception {
    // Create a parser factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Make the parser
    SAXParser saxParser = factory.newSAXParser();
    XMLReader parser = saxParser.getXMLReader();
    // Create a handler
    Handler handler = new Handler();
    // Tell the parser to use this handler
    parser.setContentHandler(handler);
    // Finally, read and parse the document
    parser.parse("hello.xml");
  }
}
```

You will need to put the file hello.xml:
  a. In the same directory, if you run the program from the command line
  b. Or where it can be found by the particular IDE you are using
Attributes

Attributes interface

- Provides access to a list of attributes present in an XML document

<table>
<thead>
<tr>
<th>Methods</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLength</td>
<td>Returns the number of attributes present in a list</td>
</tr>
<tr>
<td>getURI</td>
<td>Refers to an attribute’s Namespace URI by an index</td>
</tr>
<tr>
<td>getLocalName</td>
<td>Refers to an attribute’s local name by an index</td>
</tr>
<tr>
<td>getQName</td>
<td>Refers to an attribute’s XML 1.0 qualified name by</td>
</tr>
<tr>
<td></td>
<td>an index</td>
</tr>
<tr>
<td>getType</td>
<td>Refers to an attribute’s type by an index</td>
</tr>
<tr>
<td>getValue</td>
<td>Refers to an attribute’s value by an index</td>
</tr>
<tr>
<td>getIndex</td>
<td>Refers to the index of an attribute by a Namespace</td>
</tr>
</tbody>
</table>

Attributes interface example

```java
public static void main(String[] args) {
    try {
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser parser = spf.newSAXParser();
        parser.parse("Airlines.xml", new MyDefaultHandler());
    } catch (Exception e) { e.printStackTrace(); }
}
```

Lexical interface

- The components of lexical information are **CDATA tags, comments, and parsed entity references**
- Lexical event handling is an optional parser feature. Parser implementations are not required to support it

<table>
<thead>
<tr>
<th>Methods</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDTD</td>
<td>Describes the starting of a DTD declaration</td>
</tr>
<tr>
<td>endDTD</td>
<td>Describes the ending of a DTD declaration</td>
</tr>
<tr>
<td>startEntity</td>
<td>Describes the starting of an internal or external</td>
</tr>
<tr>
<td>endEntity</td>
<td>XML entity</td>
</tr>
<tr>
<td>startCDATA</td>
<td>Describes the starting of a CDATA section</td>
</tr>
<tr>
<td>endCDATA</td>
<td>Describes the ending of a CDATA section</td>
</tr>
</tbody>
</table>
Lexical example

```
public class LexicalExample implements LexicalHandler {

    public void startElement(String name, String publicId, String systemId) {
        throw SAXException (...);
    }
    public void endElement() throws SAXException { ... }
    public void endEntity(String name) throws SAXException { ... }
    public void startCDATA() throws SAXException { ... }
    public void endCDATA() throws SAXException { ... }
    public void comment(char ch, int start, int length) throws SAXException { ... }
    public static void main(String[] args) throws Exception {
        XMLReader reader = XMLReaderFactory.createXMLReader();
        reader.parse("MotivationDemo.xml");
    }
}
```

Locator interface

- Associates a SAX event with a particular document location
- A document location is found by the use of Locator interface
- A Locator object carries the information necessary to find the location of a document
- The Locator class encapsulates a system ID or a public identifier or both
- A default location is set by the `setDocumentLocator` before the beginning or parsing and the location of the document is found

```
public void setDocumentLocator() {
    // set default location
    ...}
```

Locator interface example

```
public class SAXLocatorExample extends DefaultHandler {

    private Locator locator;

    @Override
    public void setDocumentLocator(Locator locator) {
        this.locator = locator;
    }

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
        if (qName.equals("text")) {
            System.out.println("process element start");
        } else {
            String location = ";
            if (locator != null) {
                location = locator.getSystemId(); // XML document name
                location += " line: " + locator.getLineNumber() + " column:" + locator.getColumnNumber() + ";";
            }
            System.out.println("location: " + location);
        }
    }
}
```

Processing instruction

- Processing instructions in XML requires to be presented in a particular format
- The format for a processing instruction is `<?target data?>`

where “target” is the target application that is required to do the processing, and “data” is the instruction or information for it to process

```
public void processingInstruction(String target, String data) {
    throws SAXException {
        System.out.println(target + data);
    }
```
Entity references

- Recall that entity references are special character sequences for referring to characters that have special meaning in XML syntax
  - ‘<’ is &lt
  - ‘>’ is &gt
- In SAX these are automatically converted and passed to the characters stream unless they are part of a CDATA section

Entities & Skipped entities

- An Entity is an XML file refers to a certain piece of data or text or an entire document. SAX parser handlers these entities by the use of Entity interface
- An entity declared in an external DTD is known as skipped entity by SAX. The value of this non-declared is empty by default
- Entities in attributes are not reported rather they are skipped

UnParsed Entities example

```
<!DOCTYPE mobile [ 
  <!ELEMENT mobile (company, model, accessories)> ]> 
<!ELEMENT company (#PCDATA)> 
<!ELEMENT model (#PCDATA)> 
<!ELEMENT accessories (#PCDATA)> 
<!ENTITY ten "Nguyen Van Tu"> 
<!ENTITY test SYSTEM "explorer.exe"> 
<!ENTITY xml SYSTEM "tel.linux MDATA inj"> 
```

Errors in processing with XML

- Fatal Errors
  - Occur in the SAX parser when a document is not well formed
  - When a fatal error occurs, the parser is unable to continue
- Errors (NonFatal Errors)
  - Occurs when an XML document is not valid
  - e.g. given a DTD or schema, when a document has an invalid tag, when a tag is found where it is not allowed, or when the element contains invalid data
- Warnings
  - Generated in the presence of a DTD or schema (e.g. an element is defined twice in a DTD)
  - Generated during XML validation are not errors but the user needs to be informed about it
A non-validating SAX parser error (1)

- A non-validating SAX parser:
  - Does not validate the parser document
  - Is a general assumption that the document is valid to process
    XML continues till the end
- A parser stops parsing on intercepting an error
  - The application immediately throws an exception
  - If the application does not throw an exception, the Default Error handler generates an exception which generates a stack trace

Using non-validating parsers

A non-validating SAX parser error (2)

- Fatal Errors
  - The parser generates a SAXParseException to identify the file and location of the error and an error message is thrown
  - A SAXParserFactory class throws ParserConfigurationException for not being able to create a parser
  - If the factory class specified by the system property is not found or instantiated, a FactoryConfigurationException is thrown
  - All the Throwable objects are blocked and the remaining IOException are caught
- Non-Fatal Errors
  - The DefaultHandler method of the ErrorHandler interface is overridden
  - A SAXParseException is thrown by the SAX parser on this error
  - An exception is thrown by the DefaultHandler on finding a fatal error. Another exception would override the fatalError() method else an implementation of the SAX parser does this for it
- Warnings: are ignored since the user needs to be kept informed about the problems and their occurrences

None-valid SAX parser example

```java
public class Demo02 {
    public static void main(String[] args) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse("xml/staff.xml", new MyHandler());
        } catch (ParserConfigurationException e) {
            System.err.println("Parser Configuration Exception: " + e.getMessage());
        } catch (SAXParseException e) {
            System.err.println("SAX Parse Exception: " + e.getMessage() + " at line " + e.getLineNumber() + " column " + e.getColumnNumber());
        } catch (SAXException e) {
            System.err.println("SAX Exception: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException: " + e.getMessage());
        }
    }
}
```
None-valid SAX parser example (cont’d)

```xml
<?xml version="1.0"?>
<company>
  <staff id="1001">
    <firstname>yong</firstname>
    <lastname>mook kim
  </staff>
</company>
```

SAX Parse Exception: The element type "lastname" must be terminated by the matching end-tag "</lastname>". At line 6 - column 4

Validating SAX parser example (1)

```java
public class SAXParseException {
    public static void main (String[] args) {
        ErrorHandler error = new SAXParseException();
        try {
            XMLReader parser = XMLReaderFactory.createXMLReader();
            parser.setErrorHandler(error);
            parser.setErrorFeature("http://xml.org/sax/features/validation", true);
        } catch (Exception e) {e.printStackTrace();}
    }
}
```

Validating SAX parser example (2)

```xml
<!DOCTYPE library[]>
<library>
  <book>
    <booktitle>The Firm</booktitle>
    <author>John Grisham</author>
    <price>99</price>
  </book>
</library>
```

```
try {
    SAXParser parser = SAXParserFactory.newInstance().newSAXParser();
    parser.parse("Books.xml", new MySAXErrorHandler());
}
```
Drawbacks of SAX

- No random access possible
  - Parsing sequentially
  - Does not need to keep a copy of the whole tree structure in memory of the XML document it parsed
- Difficult to perform complex searches
  - It just traverses through the document each time there is a call to access the document
- No DTD support available: SAX 1.0 does not have any information to share about the DTD
- Cannot update documents: Basically, SAX is designed to read the XML document, not to update it
- No browser support: there are no mainstream browser available having a SAX-compliant parser

DOM (1)

- DOM is an API for HTML and XML documents
- DOM reads the entire XML document into memory and stores it as a tree data structure (with each XML element represented as a node)
- DOM acts as a language-independent interface between programs to access and modify data in a document. This provides the retrieved data in XML document a logical structure and style of representation
- Different versions: DOM 1, DOM 2, DOM 3 and DOM 4

DOM (2)

- Benefits
  - provides access to multiple documents: DOM creates the whole tree structure of XML documents while processing
  - manages complex data structures: DOM creates a structures document-tree for an XML document having complex cross references
  - allows modification of documents: it allows both reading and modifying a parsed XML document
  - allows many methods to access a document simultaneously: As the entire document is available in memory after it is parsed, you may use any of the DOM API calls multiple times to access, update, and delete document contents and structure
Working of DOM

- The DOM works with the following steps:
  - The methods of `DocumentBuilder` class in the DOM API create a tree structure of the XML document and represent each element as a node.
  - The methods contained in various interfaces in the DOM API provide access to the document and its nodes to add, modify, or delete nodes or elements in the document.

Steps to using DOM: Creating a Parsed Document

1. Import XML-related packages
   ```
   import org.w3c.dom.*;
   import javax.xml.parsers.*;
   import java.io.*;
   ```

2. Create a DocumentBuilder
   ```java
   DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = fac.newDocumentBuilder();
   ```

3. Create a Document from a file or stream
   ```java
   Document document = builder.parse(new File(file));
   ```

Steps to Using DOM: Extracting data from Parsed Document

4. Extract the root element
   ```java
   Element root = document.getDocumentElement();
   ```

5. Examine attributes
   - `getAttribute("attributeName")` returns specific attribute
   - `getAttributes()` returns a Map (table) of names/values

6. Examine sub-elements
   - `getElementsByTagName("subelementName")` returns a list of subelements of specified name
   - `getChildNodes()` returns a list of all child nodes

Steps to using DOM example

```java
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;

public class FirstDOM {
    public static void main(String[] args) throws Exception{
        DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = fac.newDocumentBuilder();
        Document doc = builder.parse("Books.xml");
        processDocument(doc);
    }
    private static void processDocument(Document doc) {
        // xml logic
    }
}
```
Document interface

- Is the root of the DOM tree, which provides access to the data in the XML document
- Contains factory methods to create the elements, text nodes, comments, and processing instructions

<table>
<thead>
<tr>
<th>Methods</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDocType</td>
<td>Returns the document type</td>
</tr>
<tr>
<td>getDocumentElement</td>
<td>Returns the attribute that allows direct access to the root element of the XML document</td>
</tr>
<tr>
<td>createElement</td>
<td>Created an element of the specified type</td>
</tr>
<tr>
<td>createTextNode</td>
<td>Creates a Text node with the string specified as the argument</td>
</tr>
<tr>
<td>createAttribute</td>
<td>Creates an Attribute in the given name. The instance of the attribute then can be set to an element using the setAttributeNode() method</td>
</tr>
<tr>
<td>getElementsByTagName</td>
<td>Returns an instance of the NodeList interface of all the elements with a given tag name in the order in which they are encountered in a pre order traversal of the document tree</td>
</tr>
</tbody>
</table>

Node interface

- Acts as the primary data type for the whole DOM model
- Contains various methods to access and manipulate the nodes in a DOM document

<table>
<thead>
<tr>
<th>Methods</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNodeName</td>
<td>Returns the name of the node depending on its type</td>
</tr>
<tr>
<td>getNodeValue</td>
<td>Returns the value of the node depending on its type</td>
</tr>
<tr>
<td>setNodeValue</td>
<td>Assigns a value to the specific node depending on its type</td>
</tr>
<tr>
<td>getChildNodes</td>
<td>Returns a NodeList containing Node’s children</td>
</tr>
<tr>
<td>getAttributes</td>
<td>Returns an instance of the NameNodeMap interface containing the attributes of the node</td>
</tr>
</tbody>
</table>

NodeList & Element interface

- NodeList Interface
  - public Node item(int index)
    - returns element at given index
      - item returns Node (parent interface of Element), so you should do a typecast on return value
  - public int getLength()
    - returns number of child elements
- Element Interface
  - The Element object is a type of node encountered in a document tree

NodeList example (1)

```java
static void printName(Document doc) throws Exception {
    NodeList nl = doc.getElementsByTagName("name");
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        System.out.println(n.getTextContent());
    }
}
```

```xml
<?xml version="1.0"?>
<Employees>
    <Employee id="1">
        <name>Pankaj</name>
        <age>29</age>
        <role>Java Developer</role>
        <gender>Male</gender>
    </Employee>
    <Employee id="2">
        <name>Lisa</name>
        <age>35</age>
        <role>CSS Developer</role>
        <gender>Female</gender>
    </Employee>
</Employees>
```

Result:
Pankaj
Lisa
**NodeList example (2)**

```xml
<?xml version="1.0" ?>
<Employees>
  <Employee id="1">
    <name>Pankaj</name>
    <age>29</age>
    <role>Java Developer</role>
    <gender>Male</gender>
  </Employee>
  <Employee id="2">
    <name>Lisa</name>
    <age>35</age>
    <role>CSS Developer</role>
    <gender>Female</gender>
  </Employee>
</Employees>
```

**Attr interface**
- Represents an attribute in an Element object
- Typically the allowable values for the attribute are defined in a schema associated with the document
- Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree

```java
Attr att = doc.createAttribute("name");
att.setValue("value");
```

**Getting Attribute example**

```java
NodeList n1 = doc.getElementsByTagName("Employee");
for (int i = 0; i < n1.getLength(); i++) {
  Node n = n1.item(i);
  System.out.println(n.getTextContent());
}
```

**Text interface**
- Inherits from CharacterData and represents the textual content (termed character data in XML) of an Element or Attr
- No lexical check is done on the content of a Text node and, depending on its position in the document, some characters must be escaped during serialization using character references:
  - e.g. the characters "&" if the textual content is part of an element or of an attribute, the character sequence "]]>" when part of an element, the quotation mark character ('') or the apostrophe character ('') when part of an attribute
Manipulating DOM

Types of Node
- Document
- DocumentFragment
- DocumentType
- ProcessingInstruction
- Text
- Comment
- CDATASection
- Element
- Attr
- EntityReference
- Entity
- Notation

Nodes in DOM tree
- Document
  - The root node of the XML document
  - The Document interface then manipulates the Document node through the methods defined in it
  - This type of node can contain only a single child node. Its child nodes can be a processing instruction element, a document type element, a comment element, elements
- DocumentFragment
  - Holds a portion of a complete document
  - Is created by the methods present in the Document interface
  - Can have processing instruction, comment, text, CDATA section, and entity reference as its child nodes
- DocumentType
  - Each document has a DOCTYPE attribute that whose value is either null or a DocumentType object
  - Provides an interface to the entities defined for the document
- ProcessingInstruction
  - This is just a processor specific instruction kept in the XML document
  - The Document interface creates a Processing Instruction node
Nodes in DOM tree

- **Element**
  - Represents an element in an XML document. Elements may contain attributes, other elements, or text. If an element contains text, the text is represented in a text-node
- **Attr**
  - Represents an attribute of an Element object. The allowable values for attributes are usually defined in a DTD
  - Because the Attr object is also a Node, it inherits the Node object's properties and methods. However, an attribute does not have a parent node and is not considered to be a child node of an element, and will return null for many of the Node properties

Nodes in DOM tree

- **Text**
  - Represents the textual content of an element or attribute
- **Comment**
  - Represents the content of comment nodes in a document
- **CDATASection**
  - Represents a CDATA section in a document
    - A CDATA section starts with "<![CDATA[" and ends with "]]>"
    - Contains text that will NOT be parsed by a parser. Tags inside a CDATA section will NOT be treated as markup and entities will not be expanded. The primary purpose is for including material such as XML fragments, without needing to escape all the delimiters
    - CDATA sections cannot be nested

Creating Nodes

- **Creating an Element Node**
  - public Element createElement(String tagName) throws DOMException
- **Creating an Attribute Node**
  - public Attr createAttribute(String Name) throws DOMException
- **Creating a Text Node**
  - public Text createTextNode(String data)
- **Creating a Comment Node**
  - public Comment createComment(String data)
- **Creating a CDATA Section Node**
  - public CDATASection createCDATASection(String data) throws DOMException
Appending Nodes

- Add a node to the end of a node list
  - public Node appendChild(Node newChild) throws DOMException

- Insert a node before a specific node
  - public Node insertBefore(Node newChild, Node refChild) throws DOMException

- Set a new attribute and attribute value
  - public void setAttribute(String name, String value) throws DOMException

- Insert data into a text node
  - public void insertData(int offset, String arg) throws DOMException

Deleting Nodes

- Remove an Element
  - element.getParentNode().removeChild(element);

- Remove an Attribute
  - element.removeAttribute("attribute name");

Modifying an Attribute

- removeAttribute
- removeAttributeNode
- setAttributeNode

```java
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = factory.newDocumentBuilder();
Document doc = docBuilder.newDocument();
Element root = doc.createElement("root");
//create child element
Element childElement = doc.createElement("Child");
//add the attribute to the child
childElement.setAttribute("attribute1", "The value of Attribute 1");
//Add text value to the child element
childElement.setTextContent("The test value");
root.appendChild(childElement);

TransformerFactory fac = TransformerFactory.newInstance();
Transformer transformer = fac.newTransformer();
transform(new DOMSource(doc), new StreamResult(System.out));
childElement.removeAttribute("attribute1");
transform(new DOMSource(doc), new StreamResult(System.out));
```
The DOM Level 2 (DOM2)

- DOM 2 modules:
  - Core
  - Event
  - Traversal
  - Range
  - Views
  - HTML
  - Style

Core Module

- Is the fundamental specification or module
- Defines a set of objects and interfaces to access and manipulate parsed XML content
- Has incorporated new ways to traverse and manipulate the XML documents through other optional modules
- Facilitates creating and populating a Document object through the DOM API calls
- Extends the functionality of the DOM core 1 with some added features

Range Module (1)

- The Range object represents a fragment of a document, or attribute
- Allows update a range of content in elements
- Can be defined as a set of values of similar type with specific boundary points
- Contains interfaces to extract a Range object from a complete document
- The content in a range is contiguous with specific boundary points

Range Module (2)

**Useful Methods in the Range Class**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloneContents()</td>
<td>Duplicates the contents of a range</td>
</tr>
<tr>
<td>deleteContents()</td>
<td>Deletes the contents of a range</td>
</tr>
<tr>
<td>getCollapsed()</td>
<td>Returns TRUE if the range is collapsed</td>
</tr>
<tr>
<td>getEndContainer()</td>
<td>getEndContainer() obtains the node within which the range ends</td>
</tr>
<tr>
<td>getStartContainer()</td>
<td>Obtains the node within which the range begins</td>
</tr>
<tr>
<td>selectNode()</td>
<td>Selects a node and its contents</td>
</tr>
<tr>
<td>selectNodeContents()</td>
<td>Selects the contents within a node</td>
</tr>
<tr>
<td>setEnd()</td>
<td>Sets the attributes describing the end of a range</td>
</tr>
<tr>
<td>setStart()</td>
<td>Sets the attributes describing the beginning of a range</td>
</tr>
</tbody>
</table>
Event Module (1)

- Design a general event system that allows registration of event handlers, defines event flow through a tree structure, and provides basic contextual information for each event.
- Develop compatibility between the current event systems used in DOM Level 0 browsers and DOM Level 2 browsers.

Event Module (2)

- Example:

```java
package dom.range;
import org.apache.xerces.dom.DocumentImpl;
import org.w3c.dom.Element;
import org.w3c.dom.Text;
import org.w3c.dom.Range;
import org.w3c.dom.Range.Template;
public class RangeDemo {
  public static void main(String[] args) {
    try {
      DocumentImpl doc = new DocumentImpl();
      Element root = doc.createElement("root");
      doc.appendChild(root);
      Text txt1 = doc.createTextNode("Đồng Văn Tình");
      Element child1 = doc.createElement("Child1");
      child1.appendChild(txt1);
      Range range = doc.createRange();
      range.selectNode(doc.getElementsByTagName("Child1").item(0));
      range.selectNode(doc.createTextNode("xxxxxx"));
      range.selectNode(doc.createTextNode("xxxxxx"));
      r2.setStart(txt1, 2); r2.setEnd(txt1, 5);
      r2.deleteContents();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Error event

```java
class MySuccessListener implements EventListener {
  public void handleEvent(Event e) {
    String s = "Event " + e.getType() + " received \n";
    s += " Event is cancelable \n";
    s += " Event is bubbling \n";
    ((Node) e.getTarget()).getNodeName() + "\n"
    System.out.println(s);
  }
}
```
Traversal Module (1)

- Allows programs and scripts to traverse through a DOM tree and identify a range of content in the document dynamically
- Allows the traversing the DOM tree to access the content in it
- Contains the TreeWalker, NodeIterator, and NodeFilter interfaces to facilitate easy traversal through the document content

Traversal Module (2)

- Using TreeWalker

```java
public TreeWalker createTreeWalker(
    Node root, int whatToShow,
    NodeFilter filter, boolean entityReferenceExpansion) throws DOMException;
```

String xmlFile = "files/employees.xml"
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(xmlFile);
Node root = doc.getFirstChild();
TreeWalker tw =
   ((DocumentTraversal)doc).createTreeWalker(
    root, NodeFilter.SHOW_ELEMENT, null, false);
Node nn = tw.getRoot();
while (nn != null)
   System.out.println(nn.getNodeName() + " 
   + nn.getNodeValue());
   nn = tw.nextNode();
```

Traversal Module (3)

- Using NodeIterator

```java
Using NodeFilter

- Facilitates the creation of the object that will filter out specific nodes present in a NodeIterator or TreeWalker
- The filter object has a user defined function to decide whether or not a node should be part of the traversal's logical view of the document
- Override the acceptNode() method that return these constants:
  - FILTER_ACCEPT: indicates that the node will be a part of the logical view of the sub-tree
  - FILTER_SKIP: indicates that the node is not a part of the logical view of the sub-tree.
    - In this case, the current node is considered as absent in the logical view, but its child not can be part of the logical view
  - FILTER_REJECT: indicates that the node and its descendants cannot be present in the logical view of the sub-tree
```

```java
String xmlFile = "files/employees.xml"
```

```java
DocumentBuilderFactory factory
    = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(xmlFile);
Node root = doc.getFirstChild();
NodeIterator iter =
   ((DocumentTraversal)doc).createNodeIterator(root,
NodeFilter.SHOW_ELEMENT, null, false);
Node n = null;
while ((n = iter.nextNode())!=null)
   System.out.println(n.getNodeName());
```
Traversal Module(4)

- Using NodeFilter

```java
NamedAnchorFilter myFilter = new NamedAnchorFilter();
NodeIterator iter =
    (DocumentTraversal)doc.createNodeIterator(
        node, NodeFilter.SHOW_ELEMENT, myFilter, false);

class NamedAnchorFilter implements NodeFilter{
    @Override
    public short acceptNode(Node n) {
        //if (n.getNodeType()==Node.ELEMENT_NODE)
        //  return FILTER_ACCEPT;
        if (!n.nodeName().equals("X")
            return FILTER_SKIP;
        if (n.getAttributeNode("NAME") != null)
            return FILTER_ACCEPT;
    }
    return FILTER_SKIP;
}
```

Other Modules

- View Module
  - Provides interfaces to facilitate the presentation of XML documents
  - Is optional in nature. Its implementation requires the implementation of the DOM 2 Core module
- Style Module
  - Provides interfaces to enable programmers to dynamically access and manipulate style sheets
  - Is optional in nature. Its implementation requires the implementation of the DOM 2 Core module
- HTML Module
  - Allows programs and scripts to access and modify the content and structure of HTML documents dynamically
  - Extends the interfaces defined in the DOM 1 HTML module

XSL

- XSL (eXtensible Stylesheet Language) is a language for expressing style sheets, consists:
  - XSLT (XSL Transformation): an XML language for transforming XML documents
  - XPath (XML Path Language): a non-XML language used by XSLT, and also available for use in non-XSLT contexts, for addressing the parts of an XML document
  - XSL-FO (XSL Formatting Objects): an XML language for specifying the visual formatting of an XML document
XML Transformations

- XML to XML: The output is an XML document
- XML to Data: The output is a byte stream document

API

<table>
<thead>
<tr>
<th>Packages</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.transform</td>
<td>Implements the generic APIs for transformation instructions and executing the transformation, starting from source to destination. The interfaces present in this package are ErrorListener, Result, Source, SourceLocator, Templates, URIResolver. The classes present in this package are OutputKeys, Transformer, TransformerFactory.</td>
</tr>
<tr>
<td>javax.xml.transform.stream</td>
<td>Implements streams and URI specific transformation APIs. The classes present in this package are StreamResult, StreamSource.</td>
</tr>
<tr>
<td>javax.xml.transform.dom</td>
<td>Implements DOM-related transformation APIs. The interface present in this package is DOMLocator.</td>
</tr>
<tr>
<td>javax.xml.transform.sax</td>
<td>Implements SAX-related transformation APIs. The interfaces present in this package are TemplateHandler, TransformerHandler.</td>
</tr>
</tbody>
</table>

XSLT Stylesheet (1)

- Is a language used for transforming XML documents into XHTML documents for Web pages
- Uses XPath language to navigate between XML documents and XHTML documents
- Defines the source documents, which will be matched with a predefined set of templates in transformation process. If XSLT gets the same document, it transforms the matching part of the source document into the output document

XSLT Stylesheet (2)

Serialization

<table>
<thead>
<tr>
<th>XML Book</th>
<th>XSLT</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>head</td>
<td>html</td>
</tr>
<tr>
<td>author</td>
<td>body</td>
<td>h1</td>
</tr>
<tr>
<td>price</td>
<td></td>
<td>h2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p</td>
</tr>
<tr>
<td></td>
<td></td>
<td>text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
XSLT Example

```
<xsl:stylesheet version="1.0" encoding="UTF-8">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>ACCP008 Excel</title>
      </head>
      <body>
        <xsl:variable name="start" select="normalize-space(.)/.">
          <p>Start: <xsl:value-of select="start"/></p>
        </xsl:variable>
        <xsl:variable name="end" select="normalize-space(.)/.">
          <p>End: <xsl:value-of select="end"/></p>
        </xsl:variable>
        <xsl:variable name="source" select="normalize-space(.)/.">
          <p>Source: <xsl:value-of select="source"/></p>
        </xsl:variable>
        <xsl:variable name="result" select="normalize-space(.)/.">
          <p>Result: <xsl:value-of select="result"/></p>
        </xsl:variable>
        <table>
          <tr>
            <th>Mã sinh viên</th>
            <th>Họ và tên</th>
            <th>Diểm thi</th>
          </tr>
          <xsl:for-each select="<xsl:variable name="result" select="normalize-space(.)/.">
            <tr>
              <td>AP0001</td>
              <td>Trần Văn An</td>
              <td>80</td>
            </tr>
            <tr>
              <td>AP0002</td>
              <td>Đặng Thị Thanh</td>
              <td>55</td>
            </tr>
            <tr>
              <td>AP0003</td>
              <td>Tăng An H平</td>
              <td>34</td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```
The object, which implements the Template interface, is the runtime result of transformation instructions.

Can be called from multiple threads for a particular instance.

Example

```java
07 public class SimpleTransform_XSLT
08 { public static void main(String[] args)
09                 throws Exception
10     { String xmlFile = "files/CustomerOrders.xml";
11       String xslFile = "files/OrderProcessing.xsl";
12       String outFile = "files/output.html";
13       StreamSource xmlSource = new StreamSource(xmlFile);
14       TransformerFactory factory = 
15           TransformerFactory.newInstance();
16       Transformer transformer = 
17           factory.newTransformer(xmlSource);
18       transformer.transform( new StreamSource(xmlFile),
19             new StreamResult(outFile));
20    }
21 }
```

You can display the output in console by using `System.out` instead of using `outfile`.

Using `transformer.setOutputProperty(OutputKeys.INDENT, "yes")` to display result with indentation.

Transform with parameter

```java
public class TransformWithParameter
```

```xml
<xs:stylesheet version="1.0" xmlns:xs="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
      <xsl:for-each select="/catalog/"/>
    </xsl:template>
</xs:stylesheet>
```
JAXP API For XPath Processing

JAXP API For XPath Processing (1)

- Define in the `javax.xml.xpath` package
- Has 2 interfaces and 2 classes:

```
JAXP API
├── XPath
│   └── XPathExpression
├── XPathFactory
└── XPathConstants
```

JAXP API For XPath Processing (2)

- The `XPath` interface gives syntax for traversing through the nodes in an XML document
- The `XPathExpression` interface deals with location path and predicates
- The `XPathFactory` class is used for creating XPath objects
- The `XPathConstants` class defines the data types such as Boolean, NodeSet, number, and string for working with nodes in and XML document