XML Processing

SDEV 4404 Advanced Software Development:
Service-Oriented Software Development

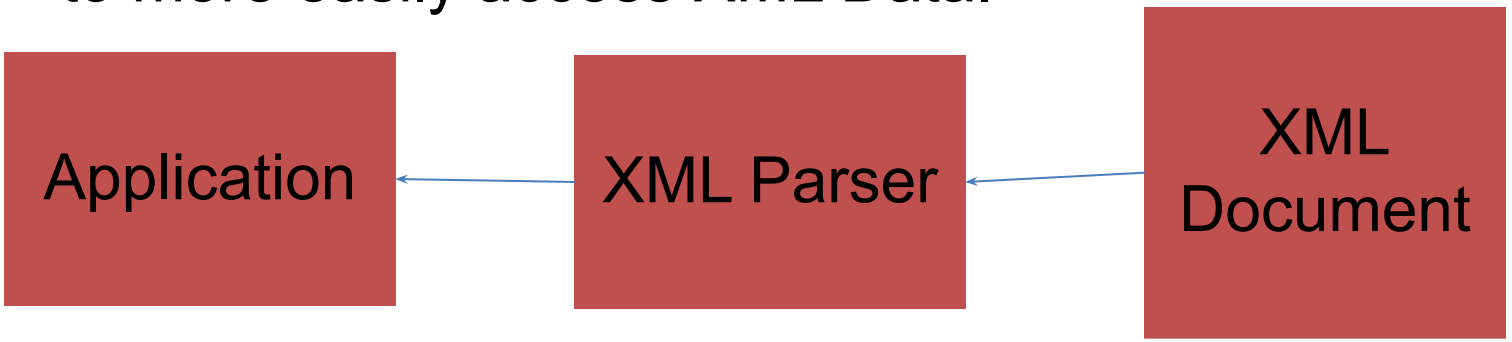
Eng. Dr. Rebhi Baraka
rbaraka@mail.iugaza.edu
Department of Software Development
Faculty of Information Technology
The Islamic University of Gaza

Outline

- Simple API for XML (SAX)
- Document Object Model (DOM)
- Streaming API for XML (StAX)
- Java API for XML Processing (JAXP)

XML Processing

- We can delete, add, or change an element (as long as the document is still valid, of course!), change its content or add, delete or change an attribute.
- An XML Parser enables your Java application or Servlet to more easily access XML Data.



Broadly, there are two types of interfaces provided by XML Parsers:

- Event-Based Interface (SAX)
- Object/Tree Interface (DOM)

Simple API for XML (SAX)

- Parse XML documents using event-based model
- Provide different APIs for accessing XML document information
- Invoke listener methods
- Passes data to application from XML document
- Better performance and less memory overhead than DOM-based parsers

Simple API for XML (SAX)

- SAX parsers read XML sequentially and do event-based parsing.
- The parser goes through the document serially and invokes callback methods on preconfigured handlers when major events occur during traversal.

Example

- Given an XML document, what kind of tree would be produced?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WEATHER SYSTEM "Weather.dtd">
<WEATHER>
    <CITY NAME="Hong Kong">
    <HI>87</HI>
    <LOW>78</LOW>
</CITY>
</WEATHER>
```

Example

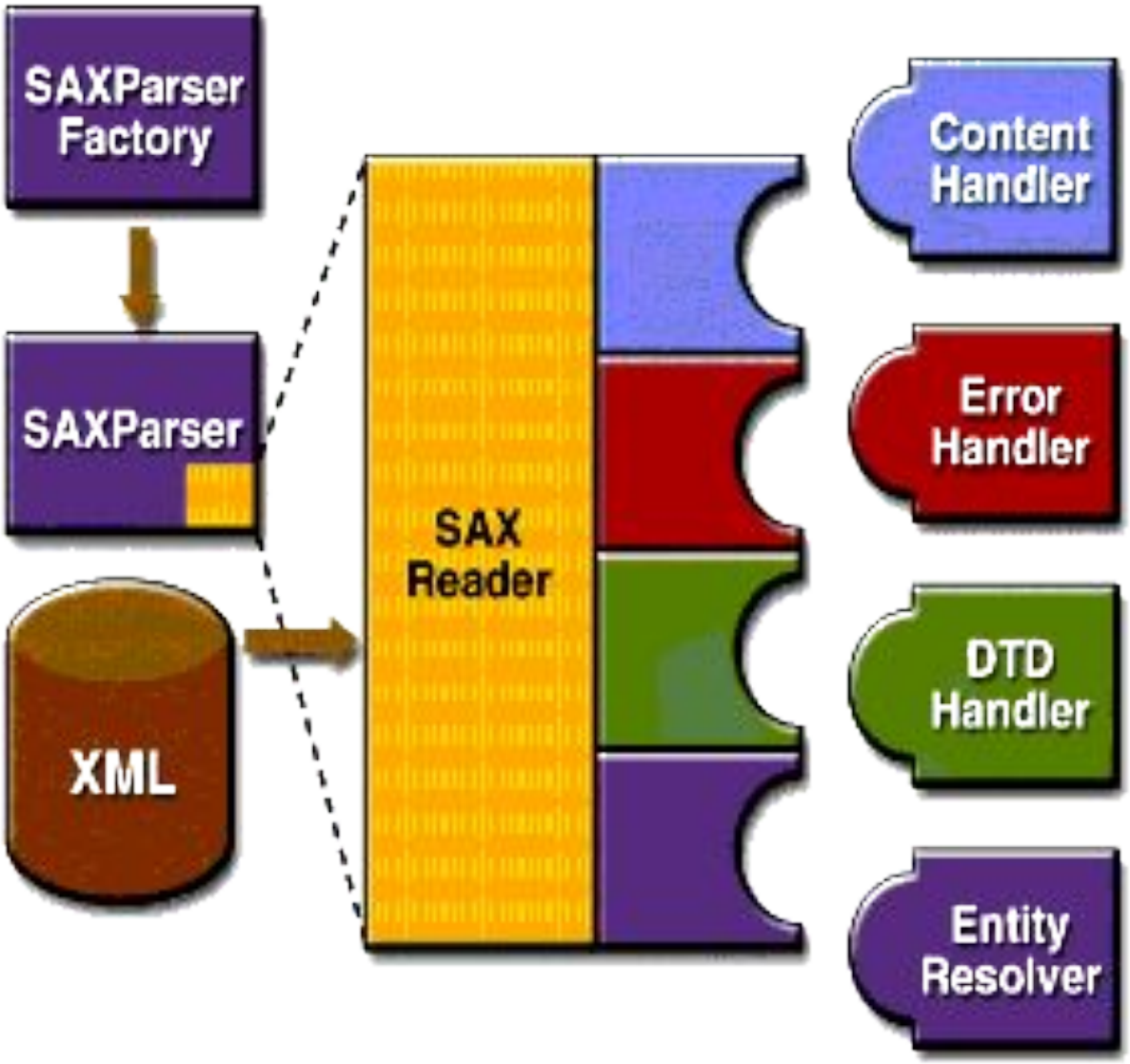
Events generated:

1. Start of <Weather> Element
2. Start of <CITY> Element
3. Start of <HI> Element
4. Character Event: 87
5. End of </HI> Element
6. Start of <LOW> Element
7. Character Event: 78
8. End of </LOW> Element
9. End of </CITY> Element
10. End of </WEATHER> Element

Event-Based Interface

- For each of these events, the application implements "event handlers."
- Each time an event occurs, a different event handler is called.
- The application intercepts these events, and handles them in any way you want.

SAX API


SAXParser Factory
SAXParser
XML
SAX Reader
Content Handler
Error Handler
DTD Handler
Entity Resolver

SAX Handlers

- The handlers invoked by the parser are :
- **org.xml.sax.ContentHandler.** Methods on the implementing class are invoked when document events occur, such as startDocument(), endDocument(),or startElement().
- **org.xml.sax.ErrorHandler.** Methods on the implementing class are invoked when parsing errors occur, such as error(), fatalError(), or warning().
- **org.xml.sax.DTDHandler.** Methods of the implementing class are invoked when a DTD is being parsed.
- **org.xml.sax.EntityResolver.** Methods of the implementing class are invoked when the SAX parser encounters an XML with a reference to an external entity (e.g., DTD or schema).

The SAX Packages

Package	Description
org.xml.sax	Defines the SAX interfaces. The name org.xml is the package prefix that was settled on by the group that defined the SAX API.
org.xml.sax.ext	Defines SAX extensions that are used for doing more sophisticated SAX processing--for example, to process a document type definition (DTD) or to see the detailed syntax for a file.

The SAX Packages

Package	Description
org.xml.sax.helpers	Contains helper classes that make it easier to use SAX--for example, by defining a default handler that has null methods for all the interfaces, so that you only need to override the ones you actually want to implement.
javax.xml.parsers	Defines the SAXParserFactory class, which returns the SAXParser. Also defines exception classes for reporting errors.

Example

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.helpers.DefaultHandler;
 public class SAXParsing {
   public static void main(String[] arg) {
      try {
      String filename = arg[0];
// Create a new factory that will create the SAX parser
      SAXParserFactory factory =
     SAXParserFactory.newInstance();
      factory.setNamespaceAware(true);
      SAXParser parser = factory.newSAXParser();
// Create a new handler to handle content
      DefaultHandler handler = new MySAXHandler();
// Parse the XML using the parser and the handler
      parser.parse(new File(filename), handler);
      } catch (Exception e) {
        System.out.println(e);
      }   }   }
```

Document Object Model (DOM)

- DOM is defined by W3C as a set of recommendations.
- The DOM core recommendations define a set of objects, each of which represents some information relevant to the XML document.
- There are also well defined relationships between these objects, to represent the document's organization.

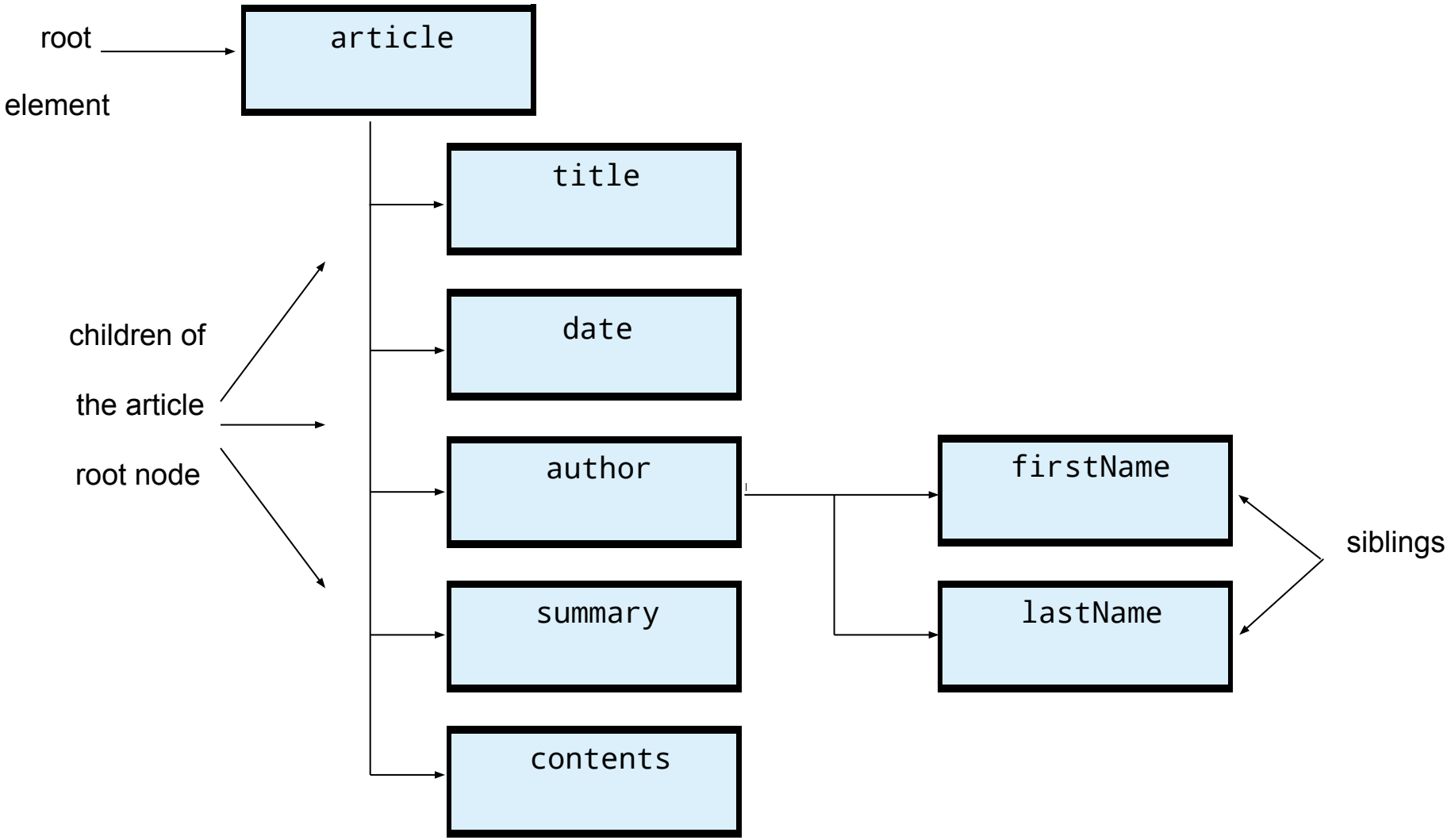
DOM Levels

- DOM is organized into levels:
  - Level 1 details the functionality and navigation of content within a document.
  - DOM Level 2 Core: Defines the basic object model to represent structured data
  - DOM Level 2 Views: Allows access and update of the representation of a DOM
  - DOM Level 2 Style: Allows access and update of style sheets
  - DOM Level 2 Traversal and Range: Allows walk through, identify, modify, and delete a range of content in the DOM
  - DOM Level 3 Working draft

Document Object Model (DOM)

- Document Object Model (DOM) tree
  - Nodes
  - Parent node
    - Ancestor nodes
  - Child node
    - Descendant nodes
    - Sibling nodes
  - One single root node
    - Contains all other nodes in document
- Application Programming Interface (API)

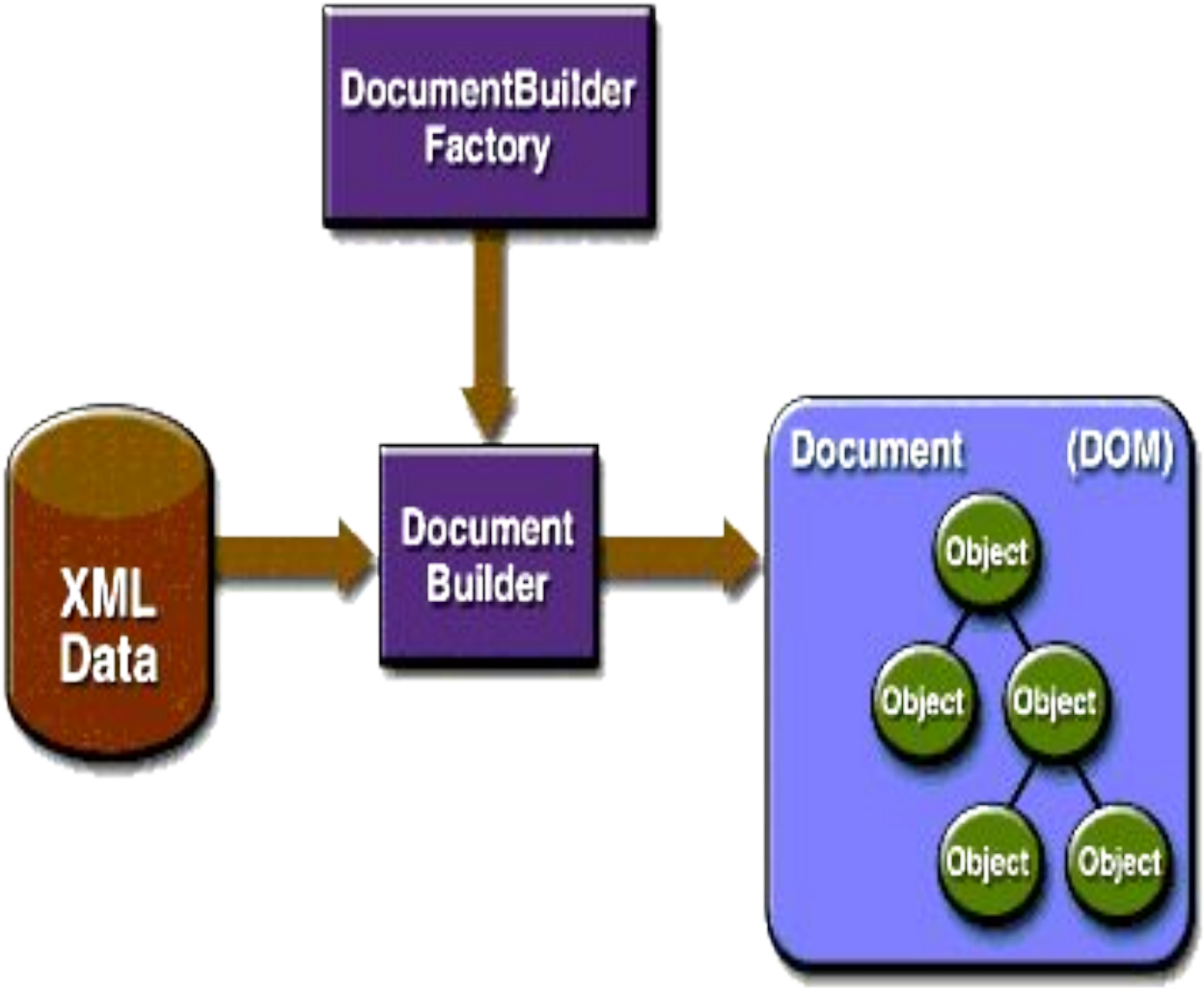
DOM tree structure for article.xml



DOM Methods

- `nodeName`
  - Name of an element, attribute, or so on
- `NodeList`
  - List of nodes
  - Can be accessed like an array using method `item`
- Property `length`
  - Returns number of children in root element
- `nextSibling`
  - Returns node's next sibling
- `nodeValue`
  - Retrieves value of text node
- `parentNode`
  - Returns node's parent node

DOM API


DocumentBuilder Factory
XML Data
Document Builder
Document (DOM)
Object
Object
Object
Object
Object

The DOM API Packages

Package	Description
org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the new Instance method. This package also defines the ParserConfigurationException class for reporting errors.

Parsing XML using DOM

```
<?xml version="1.0"?>
 <howto>
  <topic>
   <title>Java</title>
   <url>http://www.rgagnon/javahowto.htm</url>
  </topic>
  <topic>
    <title>PowerBuilder</title>
    <url>http://www.rgagnon/pbhowto.htm</url>
  </topic>
   <topic>
   <title>Javascript</title>
   <url>http://www.rgagnon/jshowto.htm</url>
  </topic>
  <topic>
   <title>VBScript</title>
   <url>http://www.rgagnon/vbshowto.htm</url>
  </topic>
 </howto>
```

howto.xml will be parsed by the program in next slide.

```
import java.io.File;
import javax.xml.parsers.*;
import org.w3c.dom.*;
public class HowtoListerDOM {
 public static void main(String[] args) {
   File file = new File("howto.xml");
   try {
     DocumentBuilder builder =
       DocumentBuilderFactory.newInstance().newDocumentBuilder();
     Document doc = builder.parse(file);

     NodeList nodes = doc.getElementsByTagName("topic");
     for (int i = 0; i < nodes.getLength(); i++) {
       Element element = (Element) nodes.item(i);

       NodeList title = element.getElementsByTagName("title");
       Element line = (Element) title.item(0);

       System.out.println("Title:"+ getCharacterDataFromElement(line));

       NodeList url = element.getElementsByTagName("url");
       line = (Element) url.item(0);
       System.out.println("Url: " + getCharacterDataFromElement(line));
     }
   }
   catch (Exception e) {e.printStackTrace();}
 }
```

```
public static String getCharacterDataFromElement(Element e)
{
   Node child = e.getFirstChild();
   if (child instanceof CharacterData) {
     CharacterData cd = (CharacterData) child;
       return cd.getData();
     }
   return "?";
 }
}
```

Title: Java Url:
http://www.rgagnon/javahowto.htm Title:
PowerBuilder Url:
http://www.rgagnon/pbhowto.htm Title:
Javascript Url:
http://www.rgagnon/jshowto.htm Title:
VBScript Url:
http://www.rgagnon/vbshowto.htm

Output of the program

When to Use What

- SAX processing is faster than DOM,
  - because it does not keep track of or build in memory trees of the document, thus consuming less memory,
  - and does not look ahead in the document to resolve node references.
  - Access is sequential, it is well suited to applications interested in reading XML data and applications that do not need to manipulate the data, such as applications that read data for rendering and applications that read configuration data defined in XML.
- Applications that need to filter XML data by adding, removing, or modifying specific elements in the data are also well suited for SAX access. The XML can be read serially and the specific element modified.

When to Use What

- Creating and manipulating DOMs is memory-intensive, and this makes DOM processing a bad choice if the XML is large and complicated or the JVM is memory-constrained, as in J2ME devices.
- The difference between SAX and DOM is the difference between sequential, read-only access and random, read-write access
- If, during processing, there is a need to move laterally between sibling elements or nested elements or to back up to a previous element processed, DOM is probably a better choice.

Streaming API for XML (StAX)

- StAX is event-driven, pull-parsing API for reading and writing XML documents.
- StAX enables you to create bidirectional XML parsers that are fast, relatively easy to program, and have a light memory footprint.
- StAX is provided in the latest API in the JAXP family (JAXP 1.4), and provides an alternative to SAX, DOM,
- Used for high-performance stream filtering, processing, and modification, particularly with low memory and limited extensibility requirements.
- Streaming models for XML processing are particularly useful when our application has strict memory limitations, as with a cellphone running J2ME, or when your application needs to simultaneously process several requests, as with an application server.

Streaming API for XML (StAX)

- *Streaming* refers to a programming model in which XML data are transmitted and parsed serially at application runtime, often from dynamic sources whose contents are not precisely known beforehand.
- stream-based parsers can start generating output immediately, and XML elements can be discarded and garbage collected immediately after they are used.
- The trade-off with stream processing is that we can only see the xml data state at one location at a time in the document.
  - We need to know what processing we want to do before reading the XML document.

Streaming API for XML (StAX)

- Pull Parsing Versus Push Parsing:
  - Streaming *pull parsing* refers to a programming model in which a client application calls methods on an XML parsing library when it needs to interact with an XML document
    - the client only gets (pulls) XML data when it explicitly asks for it.
  - Streaming *push parsing* refers to a programming model in which an XML parser sends (pushes) XML data to the client as the parser encounters elements in an XML document
    - the parser sends the data whether or not the client is ready to use it at that time.

StAX Use Cases

- **Data binding**
  - Unmarshalling an XML document
  - Marshalling an XML document
  - Parallel document processing
  - Wireless communication
- **SOAP message processing**
  - Parsing simple predictable structures
  - Parsing graph representations with forward references
  - Parsing WSDL
- **Virtual data sources**
  - Viewing as XML data stored in databases
  - Viewing data in Java objects created by XML data binding
  - Navigating a DOM tree as a stream of events

StAX API

- The StAX API is really two distinct API sets:
  - a *cursor* API represents a cursor with which you can walk an XML document from beginning to end. This cursor can point to one thing at a time, and always moves forward, never backward, usually one element at a time.
  - an *iterator* API represents an XML document stream as a set of discrete event objects. These events are pulled by the application and provided by the parser in the order in which they are read in the source XML document.

StAX API

:Examples

} public interface XMLStreamReader
;public int next() throws XMLStreamException
;public boolean hasNext() throws XMLStreamException
;()public String getText
;()public String getLocalName
;()public String getNamespaceURI
{ other methods not shown ... //

```
public interface XMLEventReader extends Iterator {
  public XMLEvent nextEvent() throws XMLStreamException;
  public boolean hasNext();
  public XMLEvent peek() throws XMLStreamException;   ... }
```

Cursor example

```
try
  {
    for(int i =  0 ; i  < count ; i++)
      {
        //pass the file name.. all  relative  entity
        //references will be resolved against  this as
        //base URI.
        XMLStreamReader xmlr  =
xmlif.createXMLStreamReader(filename, new
FileInputStream(filename));
        //when XMLStreamReader is created, it is positioned
at START_DOCUMENT event.
        int eventType = xmlr.getEventType();
        //printEventType(eventType);
        printStartDocument(xmlr);
        //check if there are  more events  in  the input stream
        while(xmlr.hasNext())
          {
            eventType =  xmlr.next();
            //printEventType(eventType);
            //these functions  prints the information about
the  particular event by calling relevant function
            printStartElement(xmlr);
            printEndElement(xmlr);
            printText(xmlr);
            printPIData(xmlr);
            printComment(xmlr);
          }
    }
```

XML Parser API Feature Summary

Feature	StAX	SAX	DOM
API Type	Pull, streaming	Push, streaming	In memory tree
Ease of Use	High	Medium	High
XPath Capability	No	No	Yes
CPU and Memory Efficiency	Good	Good	Varies
Forward Only	Yes	Yes	No
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes
Create, Read, Update, Delete	No	No	Yes

Java API for XML Processing (JAXP)

JAXP Overview

- JAXP emerged to fill in deficiencies in the SAX and DOM standards
- JAXP is an API, but more important, it is an abstraction layer.
- JAXP does not provide a new XML parsing mechanism or add to SAX, DOM or JDOM.
- It enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation.

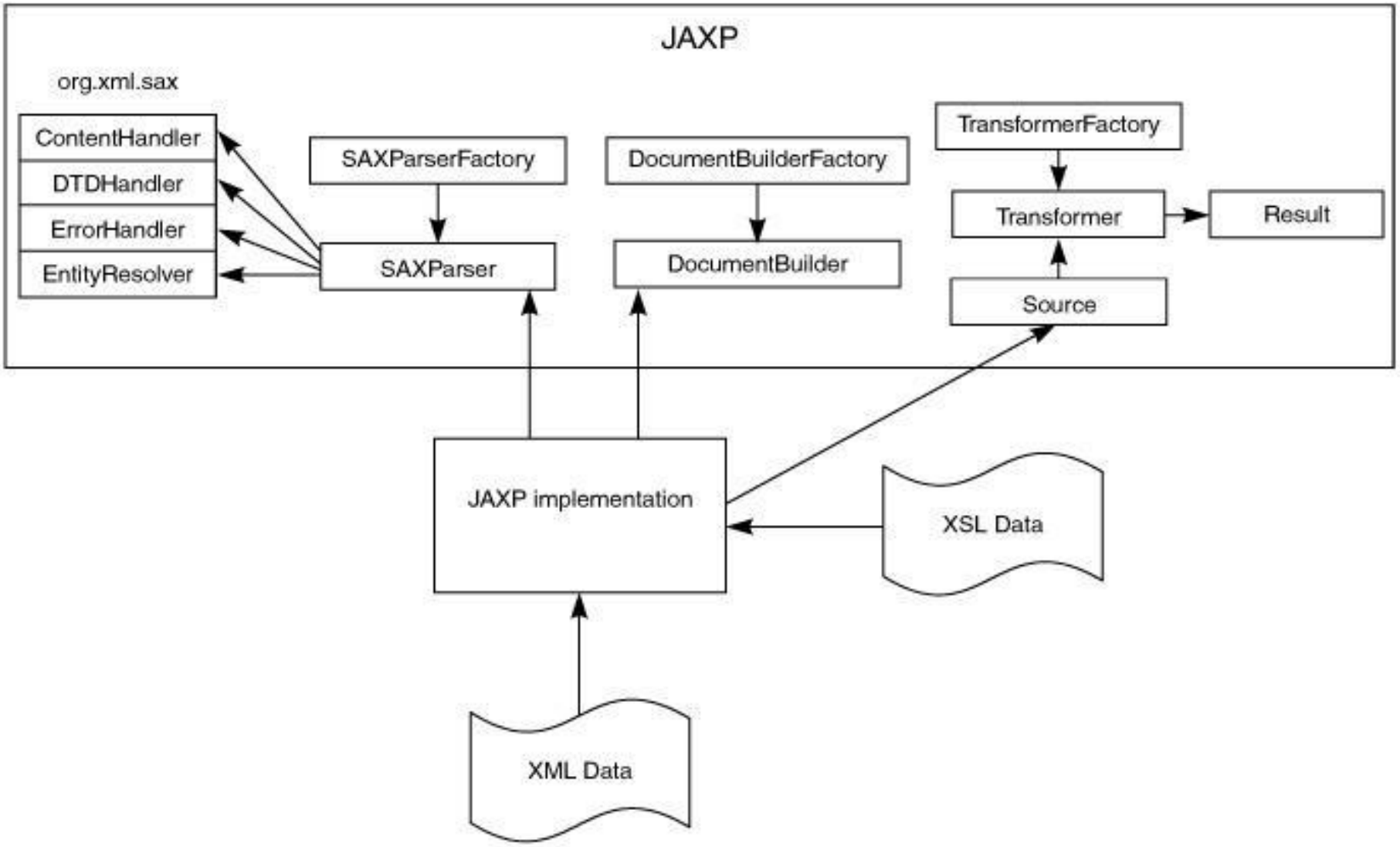
JAXP Overview

- JAXP is a standard component in the Java platform.
- An implementation of JAXP 1.4 is in Java SE 6.0.
- It supports the Streaming API for XML (StAX).

JAXP Architecture

- The abstraction in JAXP is achieved from its pluggable architecture, based on the Factory pattern.
- JAXP defines a set of factories that return the appropriate parser or transformer.
- Multiple providers can be plugged under the JAXP API as long as the providers are JAXP compliant.

JAXP Architecture



End of Slides