Parsing with the Simple API for XML (SAX)

- Let us return to the question of parsing a document.
- There are two main types of parser.
- We have briefly looked at the Document Object Model (DOM) and then in more detail at XSLT which is based on DOM.
- There is another approach to parsing:
  - SAX parsers signal events as they read the document.
  - they comply with the Simple API for XML (SAX)
- JDK 1.4 supports both SAX and DOM, as part of the Java API for XML Processing (JAXP)
SAX

- A SAX parser reads the document, once. It does not store parts that it has already read and processed in a data structure such as a tree.
- As it reads a document, it considers certain things to be "events". Examples of events are:
  - the start or end of the document
  - the start or end of an element
- You write a class (in, for example, Java) containing methods to "handle" each of the events that you are interested in, one method per event.
- Before you start the parser you register your methods with the parser.
- Then when the parser runs, every time it encounters an event, it calls your appropriate method. This is known as a callback.

SAX, continued

- A SAX parser’s "callbacks"

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A simple example of xml -->
<staff>
  <staffMember>
    <name>Robert Clark</name>
    <phone>7427</phone>
  </staffMember>
  <staffMember>
    <name>Alan Hamilton</name>
    <phone>7424</phone>
  </staffMember>
</staff>
```

(startDocument) (startElement) (characters) (endElement) (endDocument)

(ignorableWhitespace callbacks not shown)
A program that uses SAX to read XML

- Let us sketch a program that will read an XML file and echo its contents. We will look at a Java implementation.
- We will need two classes:
  1. The `XMLDemo` class containing our test method (`SAXDemo`) which will:
     1. get a parser: an instance of `XMLReader` from a "factory"
     2. register our event handlers with the parser
     3. optionally, switch on the parser to validate
     4. parse!
  2. A class containing the event handlers (`DemoSAXHandler`)

Setting things up

- The set up code in the `SAXDemo` method and in the constructor is primarily making calls of library methods and so all we are really doing is following a recipe.
- The main parts of the set up code are shown overleaf with the `try/catch` blocks missed out.
- That is one of the great things about Java. Library classes exist to carry out tasks in a large number of different application areas.
Setting things up

// Get an instance of a SAXParserFactory and get an XMLReader (a parser) from it
SAXParserFactory spFactory = SAXParserFactory.newInstance();
XMLReader reader = spFactory.newSAXParser().getXMLReader();

// Register our handler with the parser
MySAXHandler DemoSAXHandler = new DemoSAXHandler();
reader.setContentHandler(myHandler);
reader.setErrorHandler(myHandler);

// Turn off validation
reader.setFeature("http://xml.org/sax/features/validation",false);

// Associate the file we want to read
InputSource inputSource = new InputSource("staff.xml");
// and parse it..
reader.parse(inputSource);

A program that uses SAX to read XML, continued

• We now have to define class MySAXHandler, which is our event handler class
• JAXP has a DefaultHandler class that we can extend to produce DemoSAXHandler
• The events that we are interested in are:
  startDocument, characters, startElement, endElement and endDocument
• We just override the methods to provide the output for when we come across startDocument, characters, startElement, endElement and endDocument events.
Event Handlers for SAX - 1

class DemoSAXHandler extends DefaultHandler {

    // Callback when parser finds #PCDATA
    public void characters(char ch[], int start, int length) {
        System.out.print("characters callback: ");
        String s = new String(ch, start, length);
        System.out.println(s);
    } // characters

    // Callback when parser finds end of document
    public void endDocument() throws SAXException {
        System.out.println("endDocument callback");
    }

Event Handlers for SAX - 2

    // Callback when parser finds end of element
    public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
        System.out.println("endElement callback for: "+ qName);
    }

    // Callback when parser starts to read document
    public void startDocument() throws SAXException {
        System.out.println("startDocument callback");
    }

Event Handlers for SAX - 3

   // Callback when parser starts to read element
   public void startElement(String namespaceURI, String localName,
       String qName, Attributes atts) throws SAXException
   {
       System.out.println("startElement callback for: " + localName);
       // Attributes
       int nAtts = atts.getLength();
       for (int i=0 ; i<nAtts ; i++)
           System.out.println(" attribute: " + atts.getLocalName(i) +
               " is: " + atts.getValue(i));
   } //startElement

   } // end DemoSAXHandler

Running our SAX program

   • If we run this program (with Staff.xml as input) we get as output:
     startDocument callback
     startElement callback for: staff
     startElement callback for: staffMember
     startElement callback for: name
     characters callback: Robert Clark
     endElement callback for: name
     startElement callback for: phone
     characters callback: 7427
     endElement callback for: phone
     endElement callback for: staffMember
     startElement callback for: staffMember
     endElement callback for: staffMember
     ...
     endElement callback for: staffMember
     endElement callback for: staff
     endDocument callback

     - i.e. a callback for each event that we encounter
**Attributes**

- Our program will read any well-formed file
  - it will handle attributes too, for example if our XML elements had title and post attributes, we would get:

  ```
  ... 
  startElement callback for: staffMember
  startElement callback for: name
  attribute: title is: Dr
  attribute: post is: Lecturer
  characters callback: Alan Hamilton
  endElement callback for: name
  ...
  ```

**Customising our program - Comma Separated Variable (CSV)**

- That has shown us how text is output, but the actual output is not very useful.
- Let us now modify the handler so that it does something more useful.
- We will customise our program for a particular application
  - by converting Staff.xml to CSV format.
- We only need to make small changes to the event handlers as on the next slide.
Customising our program - CSV Example 1

// Characters Callback - output the text found
public void characters(char ch[], int start, int length)
{
    String s = new String(ch, start, length);
    System.out.print(s.trim());
}

// Callback when parser finds end of element
public void endElement(String namespaceURI,String localName, String qName) throws SAXException
{
    // Output a comma or a new line
    if (qName.equals("name"))
        System.out.print("," );
    else
        if (qName.equals("phone"))
            System.out.println("\n");
}

• So, given our simple XML file, this will cause the following to be output:

"Name","Phone"
"Robert Clark",7427
"Alan Hamilton",7433

Customising our program - CSV Example 2

// Callback when parser starts to read document
public void startDocument() throws SAXException
{
    System.out.println("\"Name\",\"Phone\"\n");
}

// Callback when parser starts to read element
public void startElement(String namespaceURI, String localName, String qName, Attributes atts) throws SAXException
{
    if (qName.equals("name")) System.out.print("\n");
}

• So, given our simple XML file, this will cause the following to be output:

"Name","Phone"
"Robert Clark",7427
"Alan Hamilton",7433