Managing Information (CSCU9T4)
Lecture 3: XML and Java 1 - SAX

Gabriela Ochoa

http://www.cs.stir.ac.uk/~nve/

Content

- General programming languages (Java) and XML
- SAX and DOM are corresponding APIs that are language independent and supported by numerous languages.:
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- Examples and Demos

XML AND PROGRAMMING

- XSLT, XPath and XQuery provide tools for specialised tasks.
- But many applications are not covered:
  - Domain-specific tools for concrete XML languages
  - New general tools
- We need to **parse** the XML documents (access, manipulate, create its components and structure)

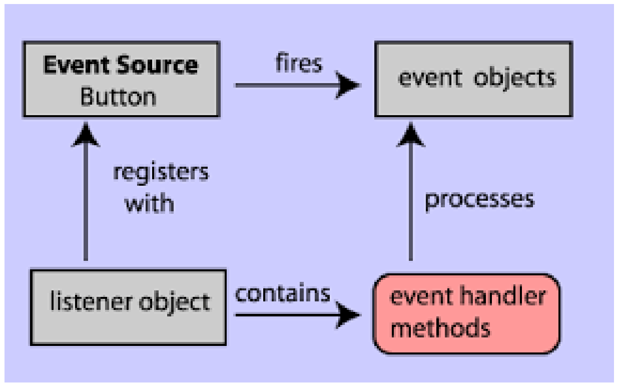
SAX and DOM APIs

- SAX and DOM are corresponding APIs that are language independent and supported by numerous languages.:
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- SAX:
  - the XML document is read step by step
  - a parser receives events as an XML document is read
  - a parser has to accept XML information as it is provided
- DOM:
  - a parser reads an entire XML document into a data structure as a tree
  - elements and attributes can then be used as required

SAX (Simple API for XML)

Event driven programming

- A programming paradigm in which the flow of the program is determined by events such as:
  - user actions (mouse clicks, key presses)
  - sensor outputs
  - messages from other programs/threads.



The most common example comes from GUI

- Objects (e.g. Buttons, TextFields, mouse, etc.) are said to generate or fire events.
- Each event source has a listener object which gets registered with the event source.
- A listener object has methods that specify what will happen when an event is sent to the listener (event handlers).
- The programmer must define the event handlers

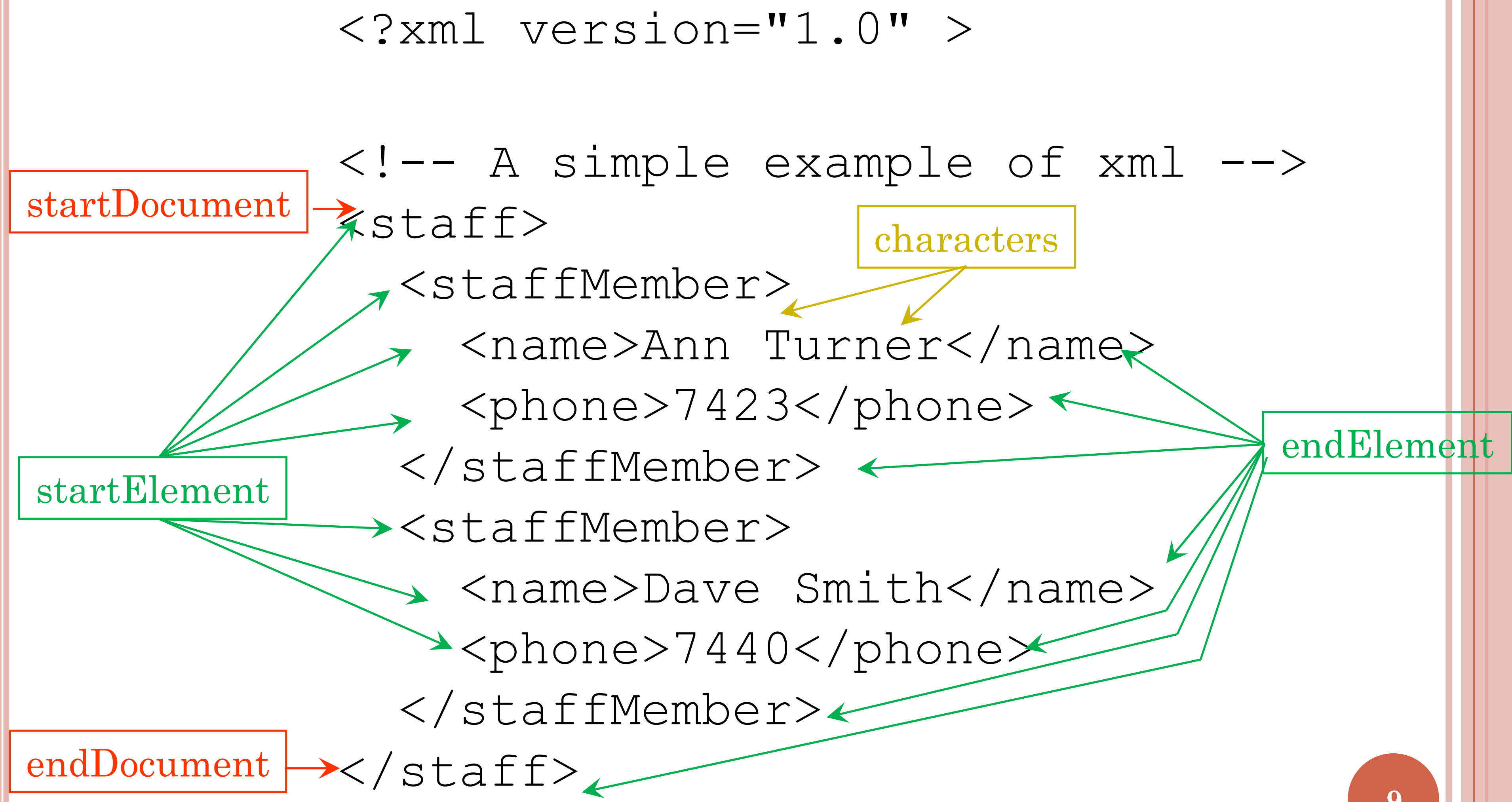
SAX Parsers

- An XML tree is not viewed as a data structure, but as a stream of events generated by the parser
- The kinds of events are, receiving:
  - the start of the document
  - the start tag of an element
  - the end tag of an element
  - character data
  - a processing instruction
  - the end of the document

SAX Parsers

- Scanning the XML file from start to end, each event invokes a corresponding **callback** method that the programmer writes.
- Note: A **callback** is some code that you pass to a given method, so that it can be called at a later time.
- Information already read and processed is no longer available
- SAX is therefore most useful for large XML documents (e.g. hundreds of thousands of elements)

SAX Events



```
<?xml version="1.0" >

<!-- A simple example of xml -->
<staff>
  <staffMember>
    <name>Ann Turner</name>
    <phone>7423</phone>
  </staffMember>
  <staffMember>
    <name>Dave Smith</name>
    <phone>7440</phone>
  </staffMember>
</staff>
```

SAX Event Handlers

- We put methods in a handler for relevant events:
  - e.g. *startDocument, startElement, endDocument*
- We then register the handler to receive events
  - When a relevant piece of XML is found, the corresponding method is given data about the event
  - An event without a handler is ignored
  - These events are named ‘callbacks’ since the system calls a method back to let it know something happened
- This is similar to (say) button handling in Java:

```
aButton.addActionListener(this);
```

  - Button events are then handled and checked in *`actionPerformed`*

Creating a Java SAX Parser

- A SAX parser needs a class with event handlers:

```
BasicHandler myHandler = new BasicHandler();
```

- Register this handler with a SAX parser/reader much as for *addActionListener* with buttons:

```
reader.setContentHandler(myHandler);
```

- Relevant events will call the defined methods:
  - Each of these corresponds to the specific types of node that are found
  - For elements, a handler still has to work out which particular element has been found (e.g. ‘name', ‘phone')

A Java SAX Example

- An example Java code that reads and prints information from an XML input file

- The *SAXDemo* class [Main class] has a method that:
  - gets a parser as an instance of *XMLReader* from a ‘factory’
  - registers event handlers with the parser
  - optionally asks the parser to validate the XML
  - handles the parsed data
- The *BasicHandler* class contains the event handlers

- The initialisation code in *SAXDemo* and its constructor calls standard library methods

- Relevant Java packages documentation
  - http://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/package-summary.html
  - http://www.saxproject.org/apidoc/org/xml/sax/package-summary.html

Program Initialisation

```
// get SAXParserFactory instance and XMLReader (i.e. parser)
SAXParserFactory spFactory = SAXParserFactory.newInstance() ;
XMLReader reader = spFactory.newSAXParser().getXMLReader();

// register handler with the parser
BasicHandler handler = new BasicHandler();
reader.setContentHandler(handler);
reader.setErrorHandler(handler);
// turn off validation
reader.setFeature("http://xml.org/sax/features/validation",false);

// associate the file to be read
InputSource inputSource = new InputSource("Staff.xml");

// parse the file
reader.parse(inputSource);
```

SAX Example

- *BasicHandler* is the event handler class:
  - It extends the *DefaultHandler* class from the Java API for XML
- The events of interest are:
  - *startDocument*
  - *characters*
  - *startElement*
  - *endElement*
  - *endDocument*

- Override default methods in the *DefaultHandler* superclass to handle events as required

OVERRIDING *DEFAULTHANDLER* METHODS

- Superclass *DefaultHandler* already contains methods called *startDocument*, *startElement*, etc.
- If the *Handler* sub-class does not provide these, the basic ones in *DefaultHandler* apply (they do nothing)
- Where a handler provides these methods, they will be called instead of the basic ones in *DefaultHandler*

Characters, Start Document Handlers

```
public class Handler extends DefaultHandler {

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

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

Start Element Handler

```
// callback when parser starts to read an element
public void startElement(String namespaceURI,
                          String localName,
                          String qName,
                          Attributes attribs) throws
SAXException
{
  System.out.println("startElement callback for " +
qName);

  // print attributes
  for (int i = 0; i < attribs.getLength(); i++)
  {
    System.out.println("  attribute " +
     attribs.getQName(i) + " is " +
attribs.getValue(i));
  }
}
```

End Element, End Document Handlers

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

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

} // end Handler
```

Output from The SAX program

- running this with Staff.xml as input gives:

```
startDocument callback
startElement callback for staff
startElement callback for staffMember
startElement callback for name
characters callback for Ann Turner
endElement callback for name
startElement callback for phone
characters callback for 7423
endElement callback for phone
endElement callback for staffMember
startElement callback for staffMember
...
endElement callback for staffMember
endElement callback for staff
endDocument callback
```

- there is thus a callback for each event

Attributes

- This program will read any well-formed XML file
- It handles attributes too
- If *staffMember* elements had a ‘post’ attribute, the output might be:

```
...
startElement callback for staffMember
startElement callback for name
attribute post is Manager
characters callback Ann Turner
endElement callback for name
...
```

Using Callbacks

- This callback example is very simple - it just prints information about what is found
  - it could print/save the information in a different format (e.g. CSV/HTML). Let's try CSV Demo.
  - the XML data could be stored in an internal data structure and then queried for particular values.
  - this could allow certain elements to be ignored.

SAX Summary

- Using SAX needs event handlers for pieces of XML that are of interest
- React to the information about each event
- Realise that XML data is provided only once
- SAX is most appropriate for very large documents or in restricted computing environments (e.g. mobile phones)