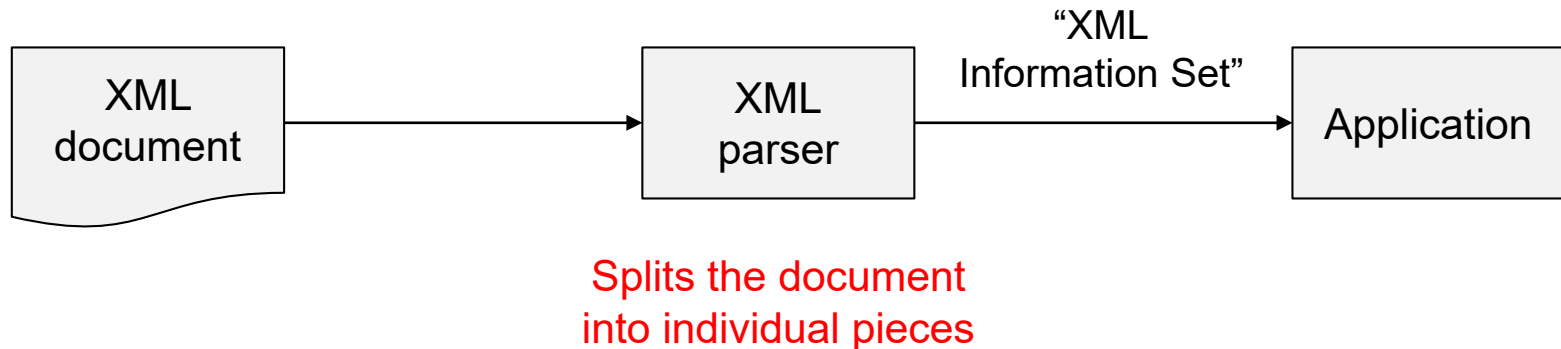


Semi-structured Data

Simple API for XML (SAX)

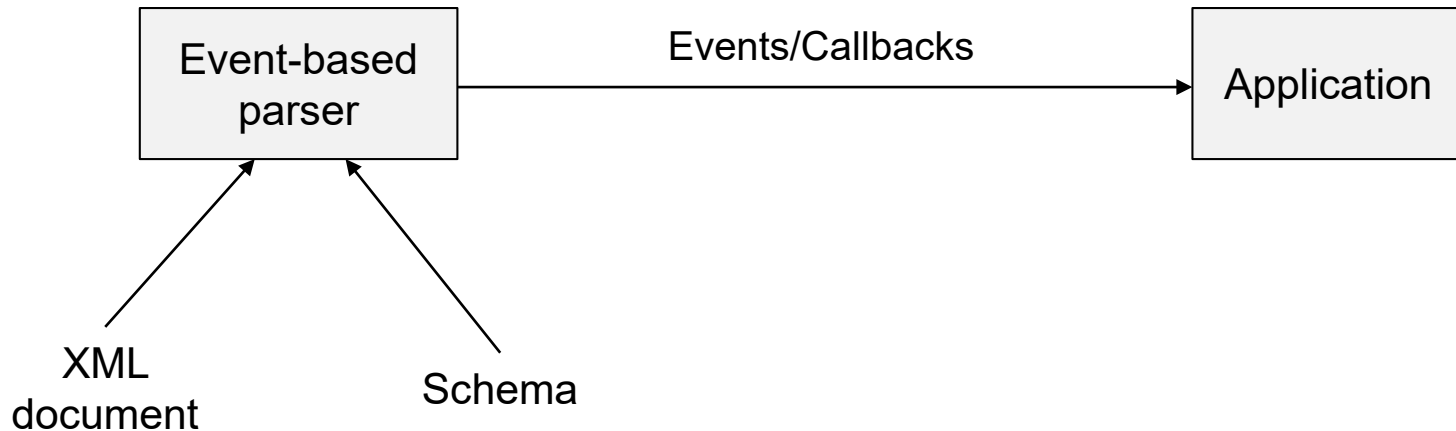
How XML Works

- Strict rules regarding the syntax of XML documents - allows for the development of **XML parsers** that can read documents
- Applications that need to understand an XML document will use a parser



Event-Based Parsers

- Report **parsing events**, such as the start and end of elements, directly to the application
- The application implements **handlers** to deal with the different events



Event-Based Parsers

parse

```
<element attr="attr-value">
  ...text-1...
  <subelement>...text-2...</subelement>
</element>
```

Events/Callbacks

start document

start element: "element"

attribute name="attr" value="attr-value"

characters: "...text-1..."

start element: "subelement"

characters: "...text-2..."

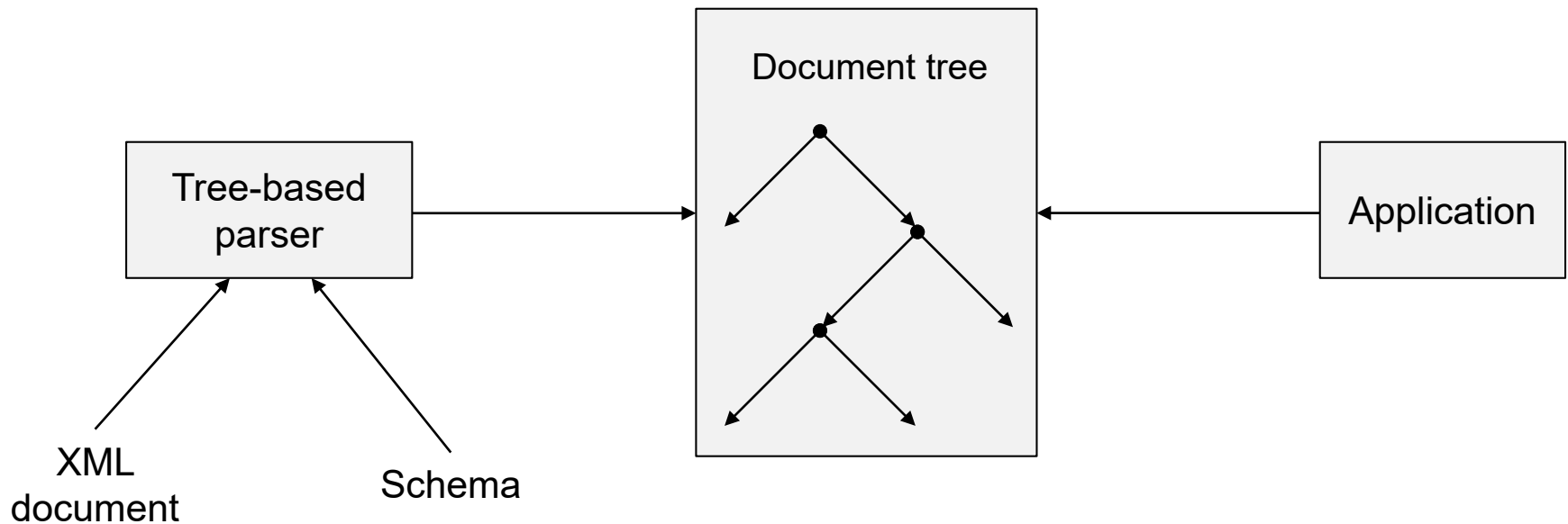
end element: "subelement"

end element: "element"

end document

Tree-Based Parsers

- Map an XML document into an **internal tree structure** stored in main memory
- The application **navigates** that tree



Tree-Based Parsers

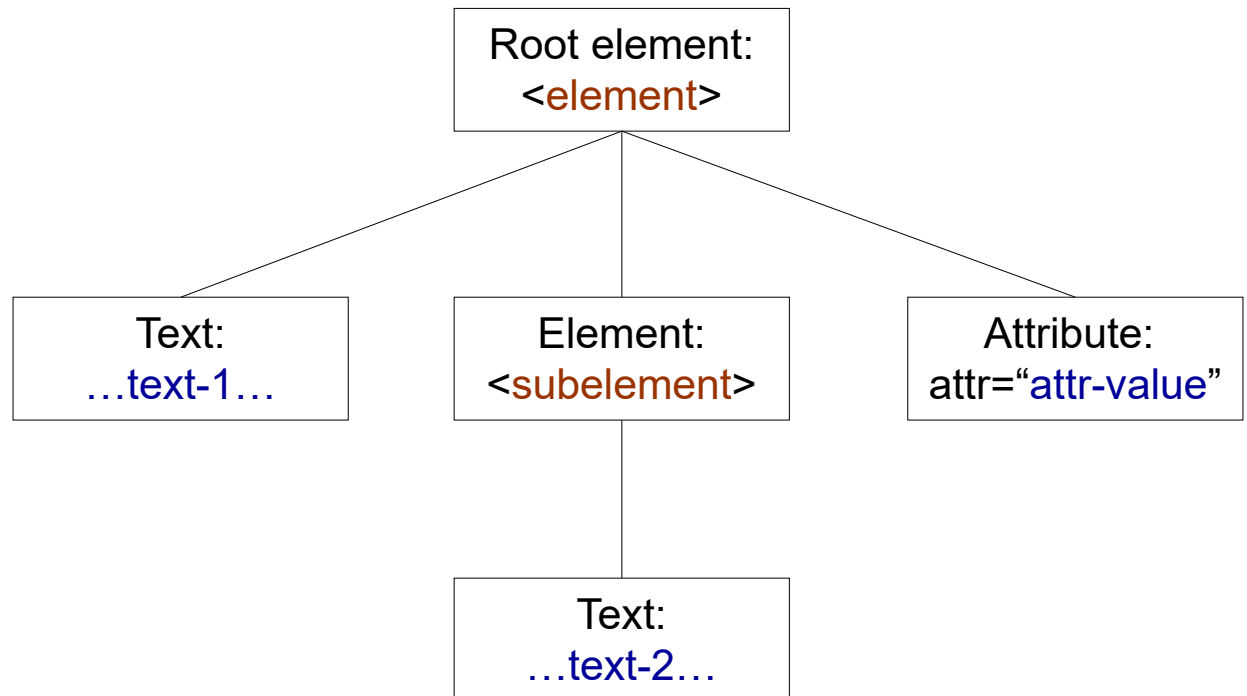
`<element attr="attr-value">`

`...text-1...`

`<subelement>...text-2...</subelement>`

`</element>`

Document Tree



Event-Based vs. Tree-Based Parsers

Event-based	Tree-based
<ul style="list-style-type: none">• Sequential access• Fast• Constant memory	<ul style="list-style-type: none">• Random access• Slow• Proportional to the document size
+	
<ul style="list-style-type: none">• Large documents• Lack of data structure	<ul style="list-style-type: none">• Small documents• Ready-made data structure

Standards for XML Parsers

- **SAX** - Simple API for XML (event-based)
 - “De facto” standard
- **DOM** - Document Object Model (tree-based)
 - W3C standard

... APIs to read and interpret XML documents

... we first focus on SAX

Outline

- Callbacks
- A Simple SAX Program
- Content Handling
- Error Handling
- Features

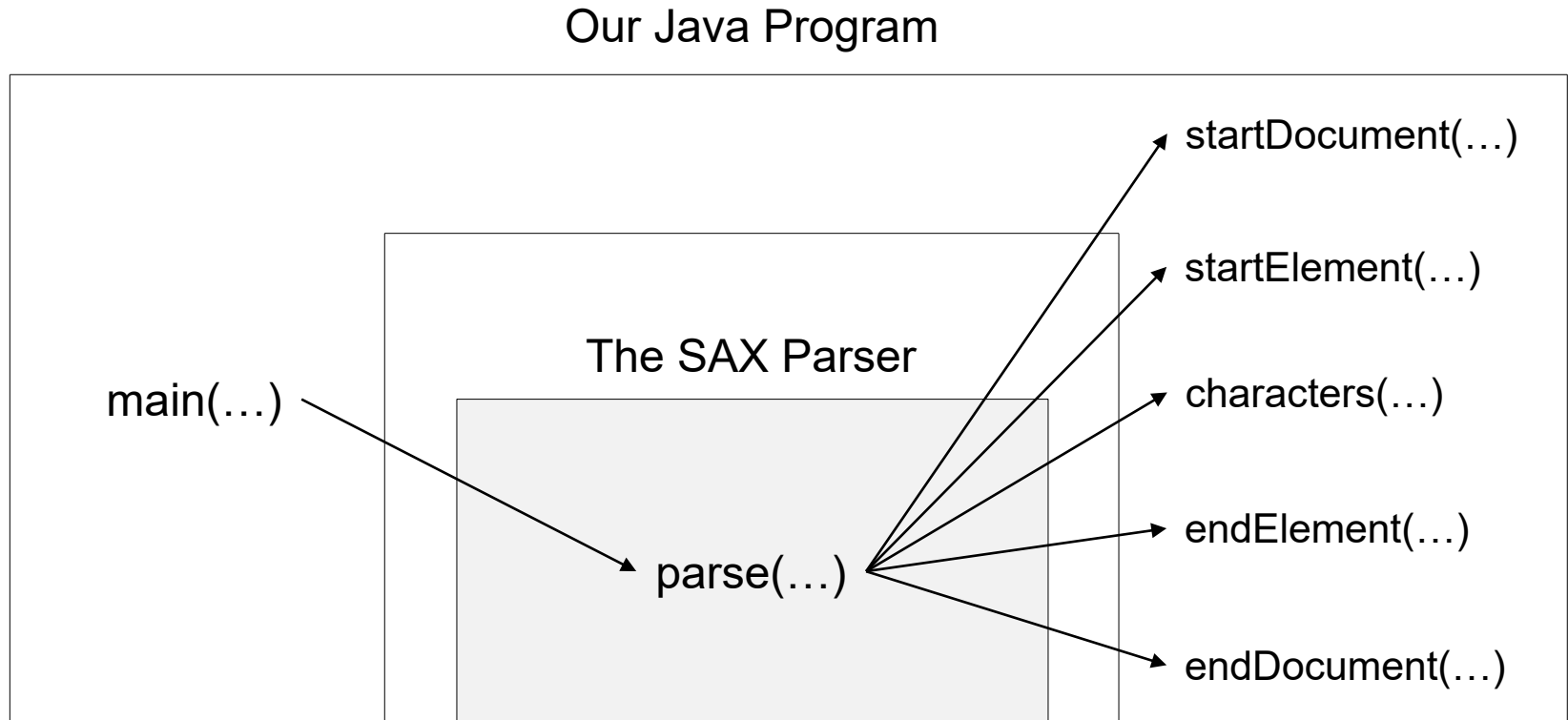
SAX - Simple API for XML

- An **event-based** API for reading XML documents
- No W3C standard, but a **“de facto” standard** - very popular
- Free and open source - <http://www.saxproject.org>
- Originally a Java-only API, but there are versions for several other programming languages (C++, Python, Perl, etc.)

ATTENTION: We focus on the Java version of the API

Callbacks

- SAX works through **callbacks** - we call the parser, it calls methods that we supply



Callbacks

- SAX works through **callbacks** - we call the parser, it calls methods that we supply
- Callback functions are divided into four **event handlers**:

- **ContentHandler** - it handles basic parsing callbacks (e.g., element starts)
- **ErrorHandler** - it handles parsing errors

- **DTDHandler** - it handles notation and unparsed entity declarations
- **EntityResolution** - customized handling for external entities

the crucial event handlers



A Simple SAX Program

course.xml

```
<?xml version="1.0"?>  
<course>Semi-structured Data</course>
```

Expected Result

```
startElement: course  
characters: Semi-structured Data  
endElement: course
```

A Simple SAX Program

- The program consists of two classes:
 - **Course** - it contains the main method
 - Creates an XMLReader - the actual parser that reads the XML document and calls the callbacks

```
XMLReader parser = XMLReaderFactory.createXMLReader();
```

- Installs the content handler

```
Handler handler = new Handler();  
parser.setContentHandler(handler);
```

- Starts the parsing

```
parser.parse("course.xml");
```

A Simple SAX Program

- The program consists of two classes:
 - **Handler** - 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 content of an element

A Simple SAX Program: Class Course

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //create XMLReader
        XMLReader parser = XMLReaderFactory.createXMLReader();

        //install the content handler
        Handler handler = new Handler();
        parser.setContentHandler(handler);

        //start parsing
        for (int i =0; i < args.length; i++) {
            parser.parse(args[i]);
        }
    }
}
```


A Simple SAX Program: Class Handler

```
import org.xml.sax.*;

public class Handler implements ContentHandler {
    //SAX calls this method when it encounters a start tag
    public void startElement(String namespaceURI,
                            String localName,
                            String qualifiedName,
                            Attributes atts) throws SAXException {
        System.out.println("startElement: " + qualifiedName);
    }
}
```

A Simple SAX Program: Class Handler

```
import org.xml.sax.*;

public class Handler implements ContentHandler {
    //SAX calls this method when it encounters a start tag
    ...

    //SAX calls this method to pass in character data
    public void characters(char[] text, int start, int length)
        throws SAXException {
        System.out.println("characters: " + new String(text, start, length));
    }
}
```

A Simple SAX Program: Class Handler

```
import org.xml.sax.*;

public class Handler implements ContentHandler {
    //SAX calls this method when it encounters a start tag
    ...

    //SAX calls this method to pass in character data
    ...

    //SAX calls this method when it encounters an end tag
    public void endElement(String namespaceURI,
                          String localName,
                          String qualifiedName) throws SAXException {
        System.out.println("endElement: " + qualifiedName);
    }
} // end of Handler class
```

A Simple SAX Program: Class Handler

```
import org.xml.sax.*;

public class Handler implements ContentHandler {
    //SAX calls this method when it encounters a start tag
    ...

    //SAX calls this method to pass in character data
    ...

    //SAX calls this method when it encounters an end tag
    ...

    //we have to implement do-nothing methods to fulfil the interface requirements
    //for example
    public void processingInstruction(String target, String data) { }
    //and several other methods
} // end of Handler class
```

A Simple SAX Program

course.xml

```
<?xml version="1.0"?>  
<course>Semi-structured Data</course>
```

Result

```
startElement: course  
characters: Semi-structured Data  
endElement: course
```

A Simple SAX Program

course.xml

```
<?xml version="1.0"?>
<course>
  <acronym>SSD</acronym>
  Semi-structured Data
</course>
```

Result

```
startElement: course
characters: \n
characters:
startElement: acronym
characters: SSD
endElement: acronym
characters:   Semi-structured Data
characters: \n
endElement: course
```

A Simple SAX Program: Class Handler

```
import org.xml.sax.*;
```

```
public class Handler implements ContentHandler {
```

```
    //SAX calls this method when it encounters a start tag
```

```
    ...
```

```
    //SAX calls this method to pass in character data
```

```
    ...
```

```
    //SAX calls this method when it encounters an end tag
```

```
    ...
```

```
    //we have to implement do-nothing methods to fulfil the interface requirements
```

```
    //for example
```

```
    public void processingInstruction(String target, String data) { }
```

```
    //and several other methods
```

```
} // end of Handler class
```

...is it possible to avoid this?

Class DefaultHandler

- In package `org.xml.sax.helpers`
- Implements all the handlers mentioned before (ContentHandler, ErrorHandler, DTDHandler, EntityResolver)
- **An adapter class** - it provides empty methods for every method declared in each of the four interfaces
- **Extend it** and override the methods that are important for the current application

<http://docs.oracle.com/javase/8/docs/api/org/xml/sax/helpers/DefaultHandler.html>

A Simple SAX Program: Class Handler Revisited

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class Handler extends DefaultHandler {
    //SAX calls this method when it encounters a start tag
    ...

    //SAX calls this method to pass in character data
    ...

    //SAX calls this method when it encounters an end tag
    ...

    //the do-nothing methods are not needed anymore
} // end of Handler class
```

Callbacks

- SAX works through **callbacks** - we call the parser, it calls methods that we supply
- Callback functions are divided into four **event handlers**:
 - **ContentHandler** - it handles basic parsing callbacks (e.g., element starts)
 - **ErrorHandler** - it handles parsing errors
 - **DTDHandler** - it handles notation and unparsed entity declarations
 - **EntityResolution** - customized handling for external entities

... more details for the methods of **ContentHandler**
can be found in the SAX-methods slides

<https://docs.oracle.com/javase/8/docs/api/org/xml/sax/ContentHandler.html>

Content Handling: Example

```
<products>
  <product>
    <name>Product A</name>
    <value>100</value>
  <product>
    <name>Product B</name>
    <value>200</value>
  <product>
    <name>Product E</name>
    <value>600</value>
  </product>
</product>
<product>
  <name>Product C</name>
  <value>400</value>
  <product>
    <name>Product E</name>
    <value>600</value>
  </product>
</product>
</product>
```

```
<product>
  <name>Product D</name>
  <value>300</value>
<product>
  <name>Product C</name>
  <value>400</value>
<product>
  <name>
    Product E
  </name>
  <value>600</value>
</product>
</product>
</product>
</products>
```

Task: Output the

- name of the product elements that are child elements of the root and
- the sum of the value elements of all of their descendants.

Content Handling: Example

```
import org.xml.sax.*;

public class TopProducts extends DefaultHandler {
    String eleText;
    private int level = 0;
    private int value = 0;

    [...]

}
```

Content Handling: Example

```
import org.xml.sax.*;

public class TopProducts extends DefaultHandler {
    String eleText;
    private int level = 0;
    private int value = 0;

    @Override
    /**
     * SAX calls this method to pass in character data
     */
    public void characters(char[] text, int start, int length)
        throws SAXException {
        eleText = new String(text, start, length);
    }
    [...]
}
```

Content Handling: Example

```
import org.xml.sax.*;

public class TopProducts extends DefaultHandler {
    String eleText;
    private int level = 0;
    private int value = 0;
    [...]

    public void startElement(String namespaceURI, String localName,
                             String qName, Attributes atts)
        throws SAXException {
        if ("product".equals(localName)) {
            level++;
        }
    }
    [...]
}
```

Content Handling: Example

```
import org.xml.sax.*;

public class TopProducts extends DefaultHandler {
    [...]
    public void endElement(String namespaceURI, String localName,
        String qName) throws SAXException {
        if ("name".equals(localName)) {
            if (level == 1) System.out.print("Total value of " + eleText + ": ");
        }
        if ("value".equals(localName)) value += Integer.parseInt(eleText);

        if ("product".equals(localName)) {
            level--;
            if (level == 0) {
                System.out.println(value);
                value = 0;
            }
        }
    }
}
```

Output:

Total value of Product A: 1900
Total value of Product D: 1300

Up to Now

- **Callbacks**
- **A Simple SAX Program**
- **Content Handling**
- Error Handling
- Features

Error Handling

- We need to install an error handler
- ... otherwise, most parsing errors will be ignored
- **ErrorHandler** - it handles parsing errors

Error Handling: Example

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //create XMLReader
        XMLReader parser = XMLReaderFactory.createXMLReader();

        //install the content and error handler
        Handler handler = new Handler();
        parser.setContentHandler(handler);
        parser.setErrorHandler(handler);

        //start parsing
        for (int i =0; i < args.length; i++) {
            parser.parse(args[i]);
        }
    }
}
```

ErrorHandler Methods

`public void fatalError(SAXParseException ex) throws SAXException`

well-formedness error

`public void error(SAXParseException ex) throws SAXException`

validation error

`public void warning(SAXParseException ex) throws SAXException`

minor error

<https://docs.oracle.com/javase/8/docs/api/org/xml/sax/ErrorHandler.html>

Error Handling: Example

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Handler extends DefaultHandler {
    //Content handling
    //Error handling
    public void fatalError(SAXParseException ex) throws SAXException {
        printError("FATAL ERROR", ex)
    }
    public void error(SAXParseException ex) throws SAXException {
        printError("ERROR", ex)
    }
    public void warning(SAXParseException ex) throws SAXException {
        printError("WARNING", ex)
    }

    private void printError(String err, SAXParseException ex) {
        System.out.printf("%s at %3d, %3d: %s \n", err, ex.getLineNumber(), ex.getColumnNumber(),
                           ex.getMessage());
    }
} // end of Handler class
```

Up to Now

- **Callbacks**
- **A Simple SAX Program**
- **Content Handling**
- **Error Handling**
- **Features**

Features

- SAX uses features to **control parser's behavior**
- Each feature has an absolute URI as a name
- Features are either true or false

Some Features

- <http://xml.org/sax/features/validation>
 - Validate the document and report validity errors
 - Default value is false
- <http://xml.org/sax/features/namespaces>
 - The parser is namespace-aware
 - Default value is true

see <https://xerces.apache.org/xerces2-j/features.html>

Example startElement Method

```
public void startElement(String namespaceURI,  
                        String localName,  
                        String qualifiedName,  
                        Attributes atts) throws SAXException
```

is called once at the **beginning of every element**

- If the parser **is namespace-aware**
 - namespaceURI holds the prefix (prefix:localname)
 - localName holds the element name (without a prefix)
 - qualifiedName might be empty
- If the parser **is not namespace-aware**
 - namespaceURI, localName might be empty
 - qualifiedName holds the element name (possibly with a prefix)

Set Feature

```
public void setFeature(java.lang.String name, boolean value)  
    throws SAXNotRecognizedException  
    throws SAXNotSupportedException
```

- **name** - the name of the feature (an absolute URI)
- **value** - value of the feature (true or false)
- **SAXNotRecognizedException** - if the feature cannot be assigned
 - Turn on validation in a non-validating parser
- **SAXNotSupportedException** - if the feature cannot be activated
 - Turn on validation (in a validating parser) when part of the document has been already parsed

Set Feature: Example

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Course {
    public static void main(String[] args) throws Exception {
        //create XMLReader
        XMLReader parser = XMLReaderFactory.createXMLReader();

        //install the content and error handler
        Handler handler = new Handler();
        parser.setContentHandler(handler);
        parser.setErrorHandler(handler);


        //turn on validation
        parser.setFeature("http://xml.org/sax/features/validation", true);

        //start parsing
        for (int i =0; i < args.length; i++) {
            parser.parse(args[i]);
        }
    }
}
```

Sum Up

- Callbacks
- A Simple SAX Program
- Content Handling
- Error Handling
- Features

Standards for XML Parsers

- **SAX** - Simple API for XML (event-based)
 - “De facto” standard 
- **DOM** - Document Object Model (tree-based)
 - W3C standard

... APIs to read and interpret XML documents

... next we will focus on DOM

Semi-structured Data

9 - Simple API for XML (SAX) Methods Overview

ContentHandler Methods

`public void startDocument() throws SAXException`

is called just once at the **beginning of parsing**

`public void endDocument() throws SAXException`

is called just once, and is the **last method called by the parser**

ContentHandler Methods

```
public void processingInstruction(String target, String data)  
    throws SAXException
```

is called once for **each processing instruction** `<?target data?>`

```
<?xml-stylesheet href="course.css" type="text/css"?>
```

target: **xml-stylesheet**

data: **href="course.css" type="text/css"**

ContentHandler Methods

```
public void startElement(String namespaceURI,  
                        String localName,  
                        String qualifiedName,  
                        Attributes atts) throws SAXException
```

is called once at the **beginning of every element**

- If the parser **is namespace-aware**
 - namespaceURI holds the prefix (prefix:localname)
 - localName holds the element name (without a prefix)
 - qualifiedName might be empty
- If the parser **is not namespace-aware**
 - namespaceURI, localName might be empty
 - qualifiedName holds the element name (possibly with a prefix)

Attributes

- When SAX calls `startElement`, it passes in a parameter of **type `Attributes`**
- `Attributes` is an interface that defines some useful methods:
 - `getLength()` - number of attributes
 - `getLocalName(index)` - attribute's local name
 - `getQName(index)` - attribute's qualified name
 - `getValue(index)` - attribute's value
 - `getType(index)` - attribute's type (CDATA, NMTOKEN, etc.)

ATTENTION: If local name is empty, then qualified name holds the attribute's name

ATTENTION: SAX does not guarantee the order of the attributes

ContentHandler Methods

```
public void endElement(String namespaceURI,  
                      String localName,  
                      String qualifiedName) throws SAXException
```

is called once at the **end of every element**

- If the parser **is namespace-aware**
 - namespaceURI holds the prefix (prefix:localname)
 - localName holds the element name (without a prefix)
 - qualifiedName is empty
- If the parser **is not namespace-aware**
 - namespaceURI, localName are empty
 - qualifiedName holds the element name (possibly with a prefix)

ContentHandler Methods

```
public void characters(char[] ch,  
                      int start,  
                      int length) throws SAXException
```

is called to pass in **character data**

```
<?xml version="1.0"?>  
<course>  
  <acronym>SSD</acronym>Semi-structured Data  
</course>
```

```
ch: <?xml version="1.0"?>  
    <course>  
        <acronym>SSD</acronym>Semi-structured Data  
    </course>
```

start: 43

SSD

length: 3

ContentHandler Methods

```
public void characters(char[] ch,  
                      int start,  
                      int length) throws SAXException
```

is called to pass in **character data**

```
<?xml version="1.0"?>  
<course>  
  <acronym>SSD</acronym>Semi-structured Data  
</course>
```

```
ch: <?xml version="1.0"?>  
    <course>  
        <acronym>SSD</acronym>Semi-structured Data  
    </course>
```

start: 56

Semi-structured Data

length: 20

ContentHandler Methods

```
public void characters(char[] ch,  
                      int start,  
                      int length) throws SAXException
```

is called to pass in **character data**

The **string constructor** can be used to extract the relevant characters

```
new String(ch, start, length)
```