XML

Working with XML in POCO.

applied informatics

Overview

- XML
- Simple API for XML (SAX)
- Document Object Model (DOM)
- Creating XML documents

XML

- eXtensible Markup Language, free, open standard
- is a general-purpose specification for creating custom markup languages
- purpose is to aid information systems in sharing structured data, expecially over the net
- documents must be well-formed (e.g. a start tag (<...>) must always be closed by an end tag (</...>)
- valid: the document can be checked against a schema (not supported by POCO)

XML – Vocabulary

- XML declaration (optional)
  <?xml version="1.0" encoding="UTF-8"?>
- root element
  the top most element starting a XML document
- elements and attributes:
  <elementName attributeName="attrValue">
  Element Content
  </elementName>
- Comments
  <!-- a comment -->

XML – Vocabulary (cont)

- Processing Instruction
  is actually information for the application. Not really of interest to the XML parser (exception: XML declaration)
  <?name data?>

- CDATA
  used to escape blocks of text containing characters which would otherwise be recognized as markup
  <xml><!CDATA[Escape <things><like></that>]]></xml>

XML Programming Interfaces

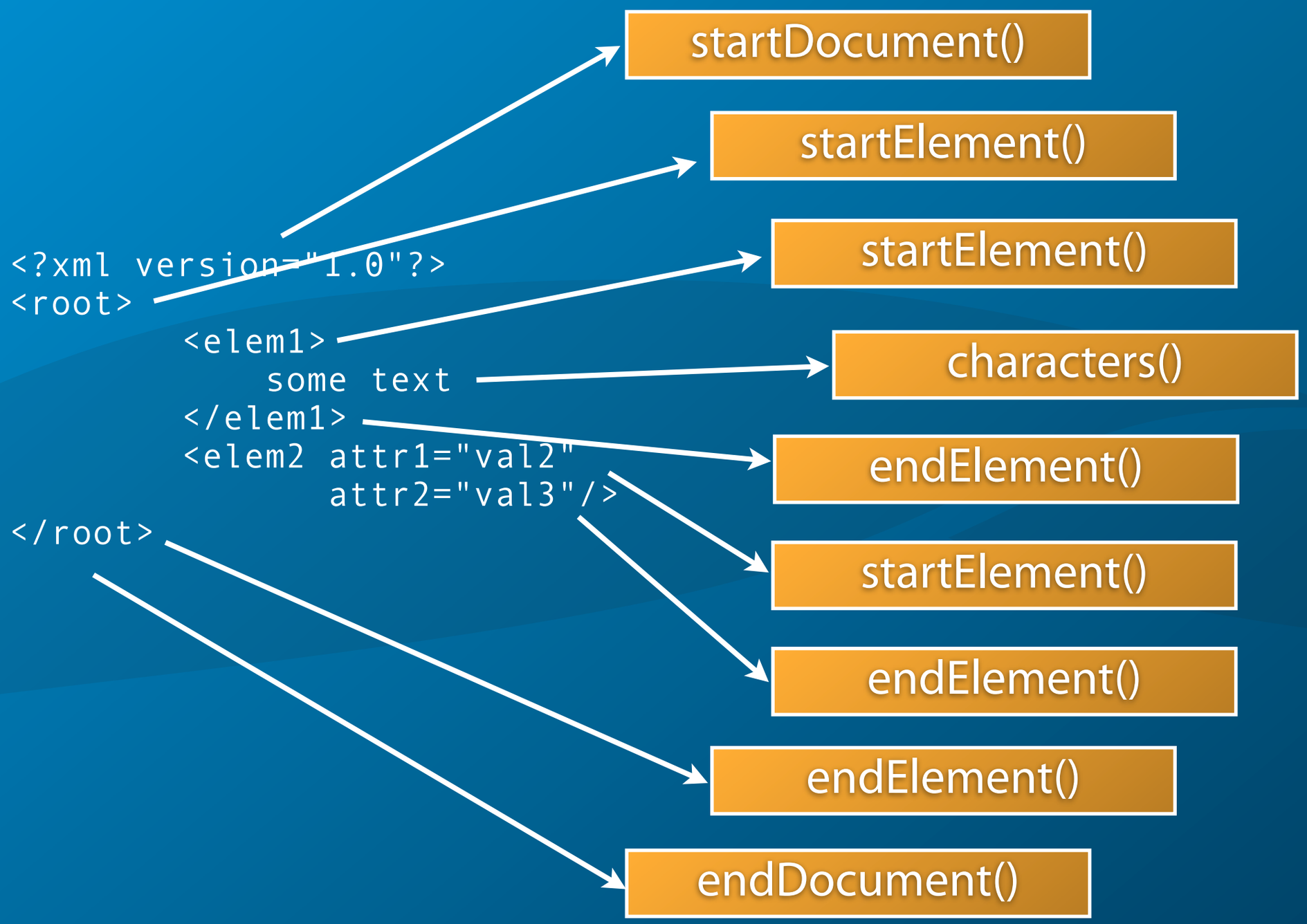
- POCO supports two interfaces for working with (reading and writing) XML data:
  - The Simple API for XML, Version 2
  - The Document Object Model

The Simple API for XML (SAX)

- SAX was originally a Java-only API for reading XML data.
- The API has been developed by a group of volunteers, not by an "official" standardization group.
- The current version of the API is 2.0.2 (since April 2004)
- POCO supports a C++ variant of the original Java API.
- For more information: http://www.saxproject.org

Event-driven Parsing

- SAX is an event-driven interface.
- The XML document is not loaded into memory as a whole for parsing.
- Instead, the parser scans the XML document, and for every XML construct (element, text, processing instruction, etc.) it finds, calls a certain member function of a handler object.
- SAX basically defines the interfaces of these handler objects, as well as the interface you use to start and configure the parser.


<?xml version="1.0"?>
<root>
<elem1>
some text
</elem1>
<elem2 attr1="val2"
attr2="val3"/>
</root>
startDocument()
startElement()
startElement()
characters()
endElement()
startElement()
endElement()
endElement()
endDocument()

SAX Interfaces

- Attributes
  (access attributes values by index or name)
- ContentHandler
  (startElement(), endElement(), characters(), ...)
- DeclHandler
  (partly supported for reporting entity declarations)
- DTDHandler
  (notationDecl(), unparsedEntityDecl())
- LexicalHandler
  (startDTD(), endDTD(), startCDATA(), endCDATA(), comment())

```
#include "Poco/SAX/ContentHandler.h"

class MyHandler: public Poco::XML::ContentHandler
{
public:
    MyHandler();

    void setDocumentLocator(const Locator* loc);

    void startDocument();

    void endDocument();

    void startElement(
        const XMLString& namespaceURI,
        const XMLString& localName,
        const XMLString& qname,
        const Attributes& attributes);

    void endElement(
        const XMLString& uri,
        const XMLString& localName,
        const XMLString& qname);
```

```
    void characters(const XMLChar ch[], int start, int length);

    void ignorableWhitespace(const XMLChar ch[], int start, int len);

    void processingInstruction(
        const XMLString& target,
        const XMLString& data);

    void startPrefixMapping(
        const XMLString& prefix,
        const XMLString& uri);

    void endPrefixMapping(const XMLString& prefix);

    void skippedEntity(const XMLString& name);
};
```

Decl Handler

- optional handler for DTD declarations in an XML file
- Document Type Declaration
- sort of simple schema language
  handles DTD declarations in an XML file

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate?, gender?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Peter Schojer</name>
    <birthdate>15/03/1976</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

```
#include “Poco/SAX/DeclHandler.h”

class MyDeclHandler: public Poco::XML::DeclHandler
{
public:
    MyDeclHandler();

    void attributeDecl(
        const XMLString& eName,
        const XMLString& aName,
        const XMLString* valueDefault,
        const XMLString* value);

    void elementDecl(const XMLString& name, const XMLString& model);

    void externalEntityDecl(
        const XMLString& name,
        const XMLString* publicId,
        const XMLString& systemId);

    void internalEntityDecl(
        const XMLString& name,
        const XMLString& value);
};
```

DTDHandler

- handles DTD not handled by DeclHandler
  - unparsed entities
  - notations
  - all reported between startDocument and first startElement

DTD Entity

- variables used to define shortcuts to text
- either as internal entity

```
<!ENTITY owner "Peter Schojer.">
<coder>&owner;</coder>
```

- or as an external entity

```
<!ENTITY owner SYSTEM “http://appinf.com/test.dtd"
<coder>&owner;</coder>
```

- predefined entities

< (<), > (>),& (&), " (“), ' (‘)

DTD Notation

- allows you to include data in your XML file which is not XML

```
<!NOTATION GIF system "image/gif">
<!ENTITY DEFAULTIMG system
  "http://appinf.com/default.gif" NDATA gif>
```

- `<image src=" &DEFAULTIMG;"/>`

```
#include “Poco/SAX/DTDHandler.h”


class MyDTDHandler: public Poco::XML::DTDHandler
{
public:
    MyDTDHandler();

    void notationDecl(
        const XMLString& name,
        const XMLString* publicId,
        const XMLString* systemId);

    void unparsedEntityDecl(
        const XMLString& name,
        const XMLString* publicId,
        const XMLString& systemId,
        const XMLString& notationName);
};
```

LexicalHandler

- optional extension handler for SAX to provide lexical information about an XML document
  - comments
  - CDATA sections
  - reports starts and end of DTD sections and entities

```
#include “Poco/SAX/LexicalHandler.h”

class MyLexicalHandler: public Poco::XML::LexicalHandler {
public:
    MyLexicalHandler();

    void startDTD(
        const XMLString& name,
        const XMLString& publicId,
        const XMLString& systemId);

    void endDTD();

    void startEntity(const XMLString& name);

    void endEntity(const XMLString& name);

    void startCDATA();

    void endCDATA();

    void comment(const XMLChar ch[], int start, int length);
};
```

SAX Parser Configuration

- XMLReader defines the interface of the parser.
- Methods for registering handlers (setContentHandler(), etc.)
- Methods for parser configuration:
  - setFeature(), getFeature()
    e.g. for enabling/disabling namespaces support
  - setProperty(), getProperty()
    e.g. for registering LexicalHandler, DeclHandler

SAX Namespaces Support

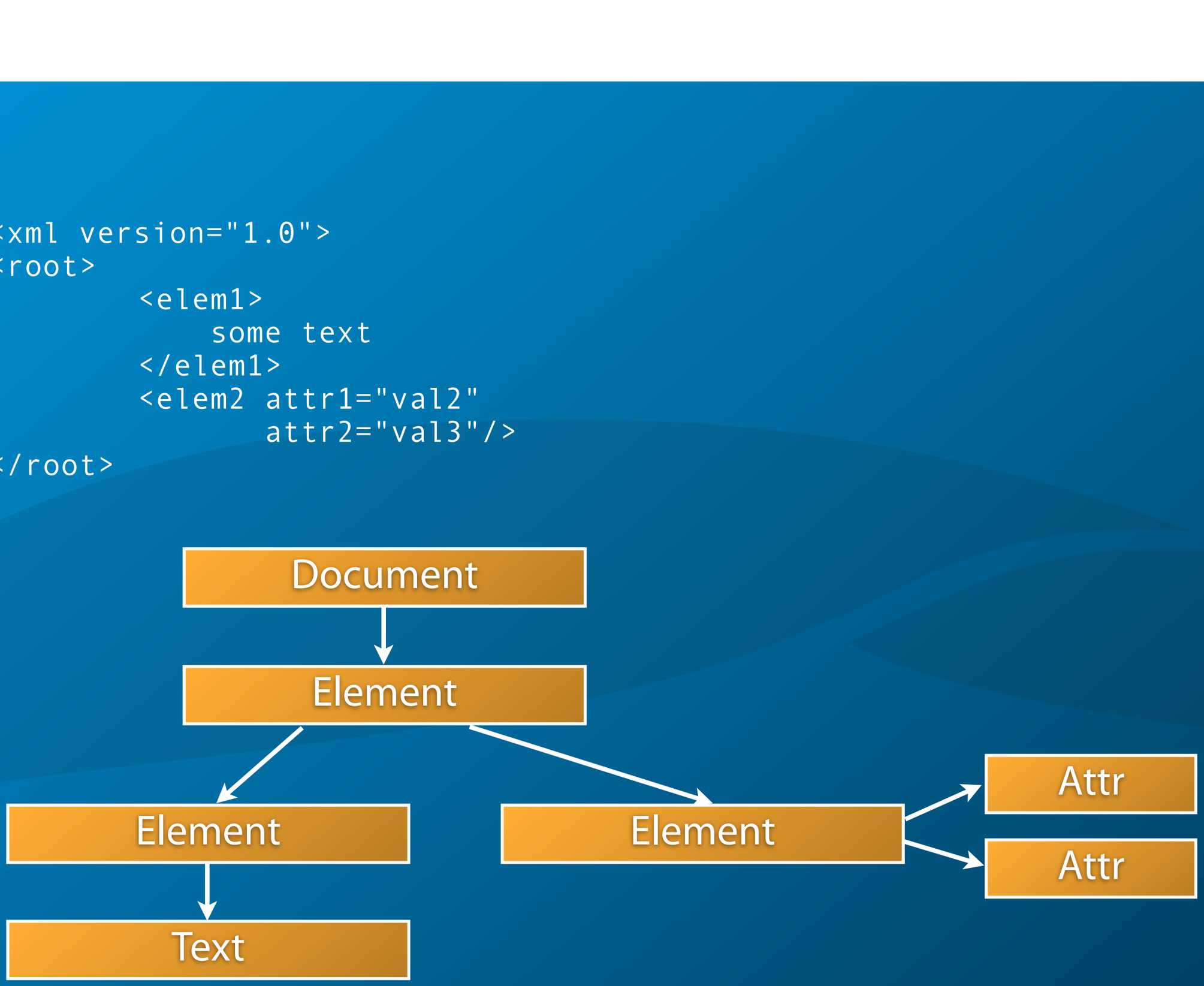
feature namespaces	feature namespace-prefixes	Namespace URI	Local Name	QName
false	false	–	–	✔
true	false	✔	✔	–
true	true	✔	✔	✔
false	true	–	–	✔

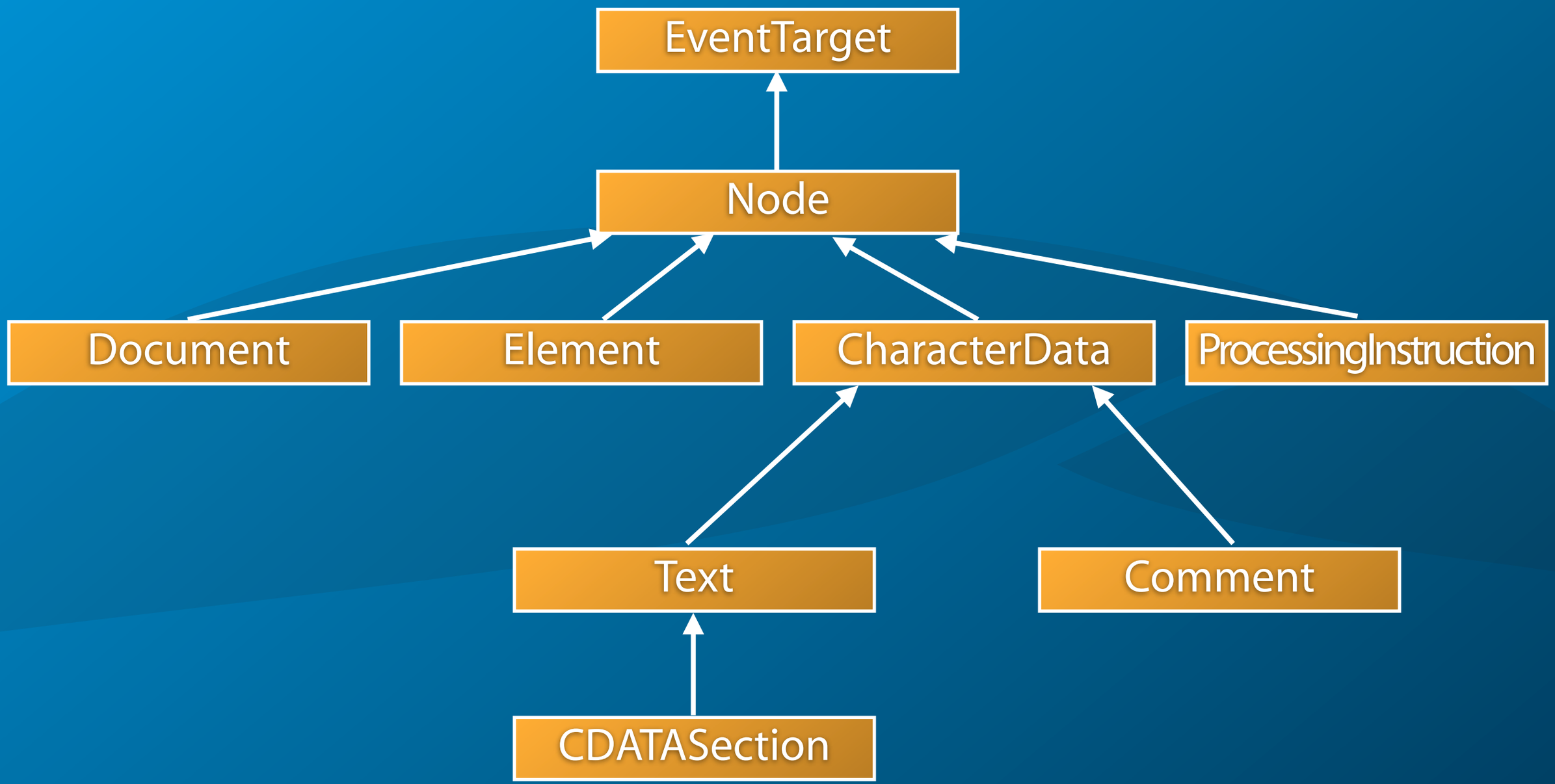
```
class MyHandler: public ContentHandler, public LexicalHandler
{
    [...]
};


MyHandler handler;
SAXParser parser;
parser.setFeature(XMLReader::FEATURE_NAMESPACES, true);
parser.setFeature(XMLReader::FEATURE_NAMESPACE_PREFIXES, true);
parser.setContentHandler(&handler);
parser.setProperty(XMLReader::PROPERTY_LEXICAL_HANDLER,
static_cast<LexicalHandler*>(&handler));

try
{
    parser.parse(“test.xml”);
}
catch (Poco::Exception& e)
{
    std::cerr << e.displayText() << std::endl;
}
```

The Document Object Model

- The Document Object Model is an API specified by the World Wide Web Consortium (W3C)
- DOM uses a tree representation of the XML document
- The entire document has to be loaded into memory
- You can modify the XML document directly

```
<xml version="1.0">
<root>
        <elem1>
            some text
        </elem1>
        <elem2 attr1="val2"
               attr2="val3"/>
</root>
```




EventTarget
Node
Document
Element
CharacterData
ProcessingInstruction
Text
Comment
CDATASection

Navigating the DOM

- Node has
  - parentNode()
  - firstChild(), lastChild()
  - nextSibling(), previousSibling()
- NodeIterator for document-order traversal:
  nextNode(), previousNode()
- TreeWalker for arbitraty navigation:
  parentNode(), firstChild(), lastChild(), etc.
- NodeIterator and TreeWalker support node filtering

Memory Management in the DOM

- DOM Nodes are reference counted.
- If you create a new node and add it to a document, the document increments its reference count. So use an AutoPtr.
- You only get ownership of non-tree objects implementing the NamedNodeMap and NodeList interface.
  You have to release them (or use an AutoPtr).
- The document keeps ownership of nodes you remove from the tree. These nodes end up in the document's AutoReleasePool.

```
#include "Poco/DOM/DOMParser.h"
#include "Poco/DOM/Document.h"
#include "Poco/DOM/NodeIterator.h"
#include "Poco/DOM/NodeFilter.h"
#include "Poco/DOM/AutoPtr.h"
#include "Poco/SAX/InputSource.h"

[...]

std::ifstream in(“test.xml”);
Poco::XML::InputSource src(in);

Poco::XML::DOMParser parser;
Poco::AutoPtr<Poco::XML::Document> pDoc = parser.parse(&src);

Poco::XML::NodeIterator it(pDoc, Poco::XML::NodeFilter::SHOW_ELEMENTS);
Poco::XML::Node* pNode = it.nextNode();
while (pNode)
{
    std::cout<<pNode->nodeName()<<":"<< pNode->nodeValue()<<std::endl;
    pNode = it.nextNode();
}
```

Creating XML Documents

- You can create an XML document by:
  - building a DOM document from scratch, or
  - by using the XMLWriter class,
  - or by generating the XML yourself.
- XMLWriter supports a SAX interface for generating XML data.

```
#include "Poco/DOM/Document.h"
#include "Poco/DOM/Element.h"
#include "Poco/DOM/Text.h"
#include "Poco/DOM/AutoPtr.h" //typedef to Poco::AutoPtr
#include "Poco/DOM/DOMWriter.h"
#include "Poco/XML/XMLWriter.h"
using namespace Poco::XML;

AutoPtr<Document> pDoc = new Document;
AutoPtr<Element> pRoot = pDoc->createElement("root");
pDoc->appendChild(pRoot);
AutoPtr<Element> pChild1 = pDoc->createElement("child1");
AutoPtr<Text> pText1 = pDoc->createTextNode("text1");
pChild1->appendChild(pText1);
pRoot->appendChild(pChild1);
AutoPtr<Element> pChild2 = pDoc->createElement("child2");
AutoPtr<Text> pText2 = pDoc->createTextNode("text2");
pChild2->appendChild(pText2);
pRoot->appendChild(pChild2);
DOMWriter writer;
writer.setNewLine("\n");
writer.setOptions(XMLWriter::PRETTY_PRINT);
writer.writeNode(std::cout, pDoc);
```

```
#include "Poco/XML/XMLWriter.h"
#include "Poco/SAX/AttributesImpl.h"


std::ofstream str(“test.xml”)
XMLWriter writer(str, XMLWriter::WRITE_XML_DECLARATION |
XMLWriter::PRETTY_PRINT);
writer.setNewLine("\n");
writer.startDocument();
AttributesImpl attrs;
attrs.addAttribute("", "", "a1", "", "v1");
attrs.addAttribute("", "", "a2", "", "v2");
writer.startElement("urn:mynamespace", "root", "", attrs);
writer.startElement("", "", "sub");
writer.endElement("", "", "sub");
writer.endElement("urn:mynamespace", "root", "");
writer.endDocument();
```

applied informatics

www.appinf.com | info@appinf.com
T +43 4253 32596 | F +43 4253 32096