XML

Advanced Programming

Extensible Markup Language- XML

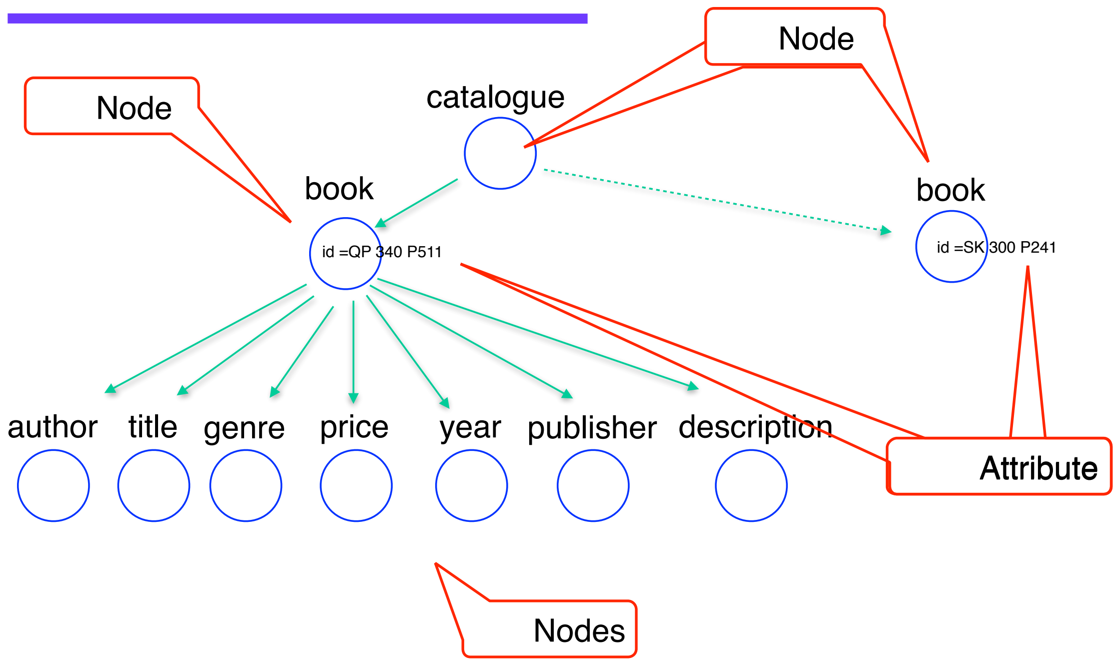
- It is a text based language which is designed to store and transport data in plain text format
- It is a tag based language like HTML, but XML tags are **not predefined** like HTML
  - You can define your own tags (i.e., extensible language)
- XML tags are designed to be **self descriptive**

XML fragments

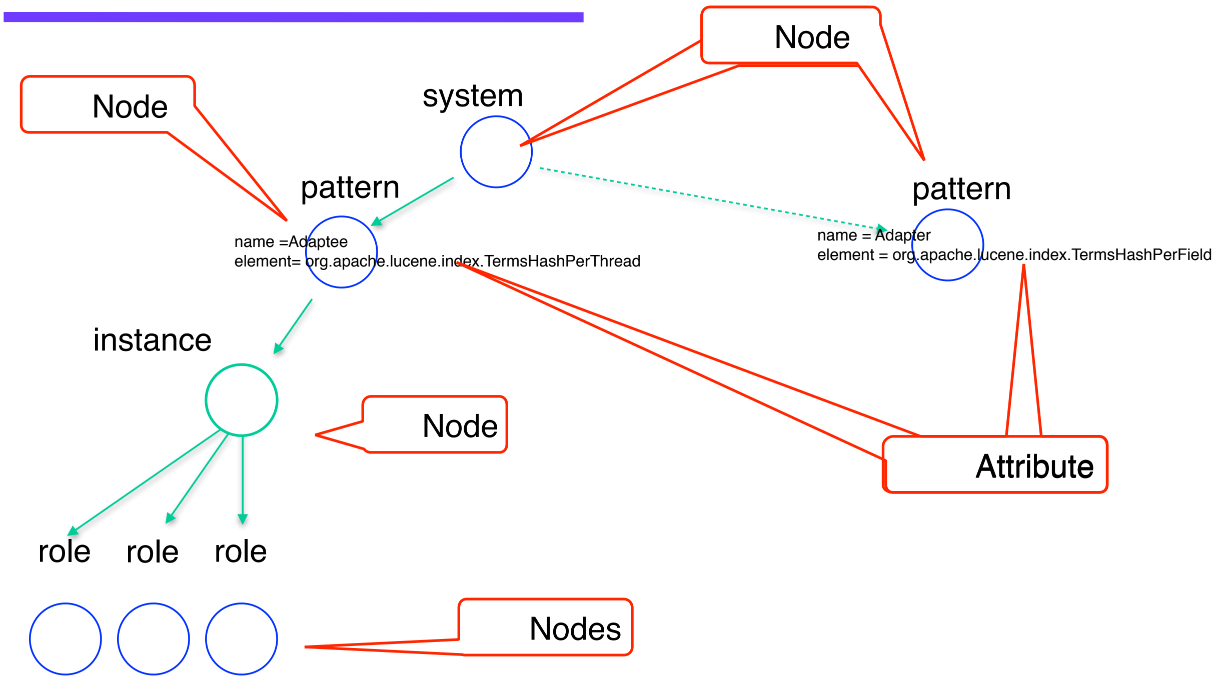
```
<?xml version="1.0" encoding="UTF-8"?>
<system>
  <pattern name="Factory Method">
       <instance>
         <role name="Adaptee" element="org.apache.lucene.index.TermsHashPerThread" />
         <role name="Adapter" element="org.apache.lucene.index.TermsHashPerField" />
         <role name="Request()" element="org.apache.lucene.index.TermsHashPerField::add():void" />
       </instance>
  </pattern>
</system>


<?xml version="1.0" encoding="UTF-8"?>
<catalogue>
  <book id="QP 340 P511">
      <author> Barbara Russo</author>
      <title>Adopting Open Source Software"</title>
      <genre> Computer Science</genre>
      <price>44.95 Euro</price>
      <year>2011</year>
      <publisher>MIT press</publisher>
      <description>A Practical Guide  to migrate to OSS in organisation</description>
  </book>
</catalogue>
```

XML tree



XML tree


Node
system
Node
pattern
pattern
name =Adaptee
element= org.apache.lucene.index.TermsHashPerThread
name = Adapter
element = org.apache.lucene.index.TermsHashPerField
instance
Node
Attribute
role
role
role
Nodes

XML documents

- They store data structures in a tree structure
- They can be queried
- They can be written/modified

Example XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<system>
	<pattern name="Factory Method">
		<instance>
			<role name="Creator" element="org.apache.lucene.analysis.Analyzer" />
			<role name="FactoryMethod()"
element="org.apache.lucene.analysis.Analyzer::tokenStream(java.lang.String,
java.io.Reader):org.apache.lucene.analysis.TokenStream" />
		</instance>
		<instance>
			<role name="Creator" element="org.apache.lucene.index.DocConsumer" />
			<role name="FactoryMethod()"
element="org.apache.lucene.index.DocConsumer::addThread(org.apache.lucene.index.DocumentsWriterThreadS
tate):org.apache.lucene.index.DocConsumerPerThread" />
		</instance>
….
		<instance>
			<role name="Creator" element="org.apache.lucene.index.DocFieldConsumerPerThread" />
			<role name="FactoryMethod()"
element="org.apache.lucene.index.DocFieldConsumerPerThread::addField(org.apache.lucene.index.FieldInfo
):org.apache.lucene.index.DocFieldConsumerPerField" />
		</instance>
	</pattern>
</system>
```

Parsing XML

- Parsing XML: going through XML document to access data or to modify data in one or other way
- Parsing XML document with a DOM parser
- What is DOM?

Major XML parsers

- **Dom Parser** - it loads the whole document and creates its hierarchical tree in memory
- **SAX Parser** - it does not load the complete document into the memory as it parses the document on event based triggers
- **XPath Parser** - It parses the XML based on expressions

The DOM standard

- The Document Object Model is an official recommendation of the World Wide Web Consortium (W3C). It is:
- An interface that enables programs to access and update the style, structure, and contents of XML docs
- XML parsers use DOM

Major elements DOM

- **Node** - The base datatype of the DOM
- **Element** - subnode
- **Attr** - an attribute of an element
- **Text** - actual content of an Element or Attr
- **Document** - the entire XML document

DOM parsing

- DOM parsing returns a tree structure that contains all of the Elements of a document
- DOM provides methods to examine the content and structure of a document

Parsing XML Document with DOM

- Import XML-related packages
- Create a DocumentBuilder
- Create a Document from a file or stream
- Extract the root element
- Examine attributes
- Examine sub-elements

Import XML-related packages

- **import org.w3c.dom.*;**
- **import javax.xml.parsers.*;**
- **javax.xml.xpath.*;**
- **org.xml.sax.*;**
- java.nio.*;
- java.util.regex.*;
- import java.io.*;

org.w3c.dom package

- Provides the interfaces for DOM
- Major interfaces we are going to use:
    - Attr
    - Node
    - Element
    - NodeList
    - Document
    - NamedNodeMap //collections of nodes

javax.xml.parsers package

- Provides classes allowing the processing of XML documents
- Classes:
    - DocumentBuilder
    - DocumentBuilderFactory
    - SAXParser
    - SAXParserFActory

javax.xml.xpath package

- The XPath language provides syntax for selecting nodes from an XML document with reg expression
- Major interfaces we are going to use:
  - XPath
  - XPathExpression
- Major classes
  - XPathFactory

Create a DocumentBuilder

- It is responsible to create DOM objects
- It is created by an object of the **Factory** abstract class “DocumentBuilderFactory” with the **Factory Method** “newDocumentBuilder()”

DocumentBuilderFactory is an abstract class with protected constructor

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();
```

Factory pattern

- First create an object of the Factory with a Factory method and static call

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

- Through the object of the Factory create a Builder object class with a Factory method

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

- The creation of an object of Document is delegated to the Builder object

We’ve already seen it …

```
Pattern comma = Pattern.compile(",");
Matcher matchingComma = comma.matcher(“Barbara,Russo");

Matcher objects have methods to search and return

boolean answer = matchingComma.matches();
String[] token = answer.find();
```

Create Document from a file/stream

```
StringBuilder xmlStringBuilder = new StringBuilder();

xmlStringBuilder.append("<?xml version="1.0"?> <class> </class>");

ByteArrayInputStream input =  new ByteArrayInputStream(

xmlStringBuilder.toString().getBytes("UTF-8"));

Document doc = builder.parse(input);

or

String XMLfilePath = “Lucene300DesignPatterns.xml";
Document doc = builder.parse(XMLfilePath);

or
Document doc =builder.parse("Lucene300DesignPatterns.xml");
```

Extract the root element

- Get the root node of the DOM tree

```
Element root = doc.getDocumentElement();
doc.getDocumentElement().normalize();
```

- Print the root node name (in the fragments is “system” or “catalogue”)

```
System.out.println("Root element: " +
doc.getDocumentElement().getNodeName());
```

Examine attributes

- It returns specific attribute

- getAttribute()

```
doc.getDocumentElement().getAttribute()
```

- It returns a Map (table) of names/values
- getAttributes()

Examine sub-elements

- NodeList is an ordered collection of nodes of the org.w3c.dom package
- It returns a list of subelements of specified name:
- getElementsByTagName("subelementName");

```
NodeList patternsList =
doc.getElementsByTagName("pattern");
```

- It contains all children of this node

```
NodeList patternsList = node.getChildNodes();
```

Using reg expressions

```
XPathFactory xPathfactory = XPathFactory.newInstance();

XPath xpath = xPathfactory.newXPath();


XPathExpression expr = xpath.compile("/system//
pattern[@name='"+patternName+"']//instance//role[@name='"+roleName
+"']");

NodeList nodeList =
(NodeList)expr.evaluate(doc,XPathConstants.NODESET);
```

Using Factory/F. method/ Builder

- First create a Factory object

```
XPathFactory xPathfactory = XPathFactory.newInstance();
```

- Then create a Builder object that builds the path in the node tree

```
XPath xpath = xPathfactory.newXPath();
```

Using Factory/F. method/ Builder

- Then compile the expression onto the path

```
XPathExpression expr = xpath.compile("/system//
pattern[@name='"+patternName+"']//instance//
role[@name='"+roleName+"']");
```

- Finally evaluate the regular expression in the document

```
NodeList nodeList =
(NodeList)expr.evaluate(doc,XPathConstants.NODESET);
```

XPath language

- The XPath language provides a syntax for selecting nodes from an XML document
- The XPath is an official recommendation of W3C
- It is used to traverse elements and attributes of an XML document
- XPath provides various types of expressions which can be used to enquire relevant information from the XML document

Examples on fragments

```
XPathExpression expr = xpath.compile(“/system//
pattern[@name='Factory Method']//instance//
role[@name='Adaptee']");

XPathExpression expr = xpath.compile("/system//
pattern[@name='Prototype']//instance//
role[@element='org.apache.lucene.search.spans.SpanQue
ry']");
```

- Hierarchy as in a file system
- /system/pattern or /catalogue/book
- indicates to position on the node at “pattern” or at “book”
- The symbol // indicates all the child nodes
- [] enclosing the type of match
- @ the attribute to match

Evaluating expressions

```
NodeList nodeList =
(NodeList)expr.evaluate(doc,XPathConstants.NODESET);
```

The type of list returned is specified by the constants in the evaluation method

List of nodes
`XPathConstants.NODESET`

Single node
`XPathConstants.NODE`

```
Node nodeList =
(Node)expr.evaluate(doc,XPathConstants.NODE);
```

Going up to the tree

```
NamedNodeMap classList = nodeList.item(i).getAttributes();

NamedNodeMap parent =
nodeList.item(i).getParentNode().getParentNode().getAttribu
tes();

String line = new String(parent.item(0).getNodeValue()
+":"+classList.item(0).getNodeValue()+'\r');

//use regular expressions
Pattern column = Pattern.compile(":");
String[] tokens= column.split(line);

System.out.println(tokens[0]+","+tokens[1]+'\r');
```

Create the XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<system>
    <pattern name="Factory Method">
        <instance name="Creator">org.apache.lucene.analysis.Analyzer
        </instance>
      <instance name="Adaptor">org.apache.lucene.analysis.Synthesis
        </instance>
    <pattern>
<system>
```

Build the document and root node

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.newDocument();

// root element

Element system = doc.createElement("system");
doc.appendChild(root);
//  pattern element
Element pattern = doc.createElement("pattern");
rootElement.appendChild(pattern);

// setting attribute to element
Attr attr = doc.createAttribute("name");
attr.setValue("Factory Method");
pattern.setAttributeNode(attr);
```

Create subnodes

```
Element instance1 = doc.createElement("instance");
rootElement.appendChild(instance);
// setting attribute to element
Attr attr = doc.createAttribute("name");
attr.setValue("Creator");
instance.setAttributeNode(attr);
instance.appendChild(doc.createTextNode("org.apache.lucene.analysis.Ana
lyzer"));
pattern.appendChild(instance);


Element instance2 = doc.createElement("instance");
rootElement.appendChild(instance);
Attr attr = doc.createAttribute("name");
attr.setValue("Adaptor");
instance.setAttributeNode(attr);
instance.appendChild(doc.createTextNode("org.apache.lucene.analysis.Ana
lyzer"));
pattern.appendChild(instance);
```

Write the content into xml file

```
TransformerFactory transformerFactory =
TransformerFactory.newInstance();
Transformer transformer =
transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new File(“myFolder/
system.xml”));
transformer.transform(source, result);
// Output to console for testing
StreamResult consoleResult = new StreamResult(System.out);
transformer.transform(source, consoleResult);
```

Modify an XML file

```
NodeList list = pattern.getChildNodes();
Node node = list.item(5);
Element eElement = (Element) node;
String text = eElement.getTextContent()
eElement.setTextContent("Adaptee");
```