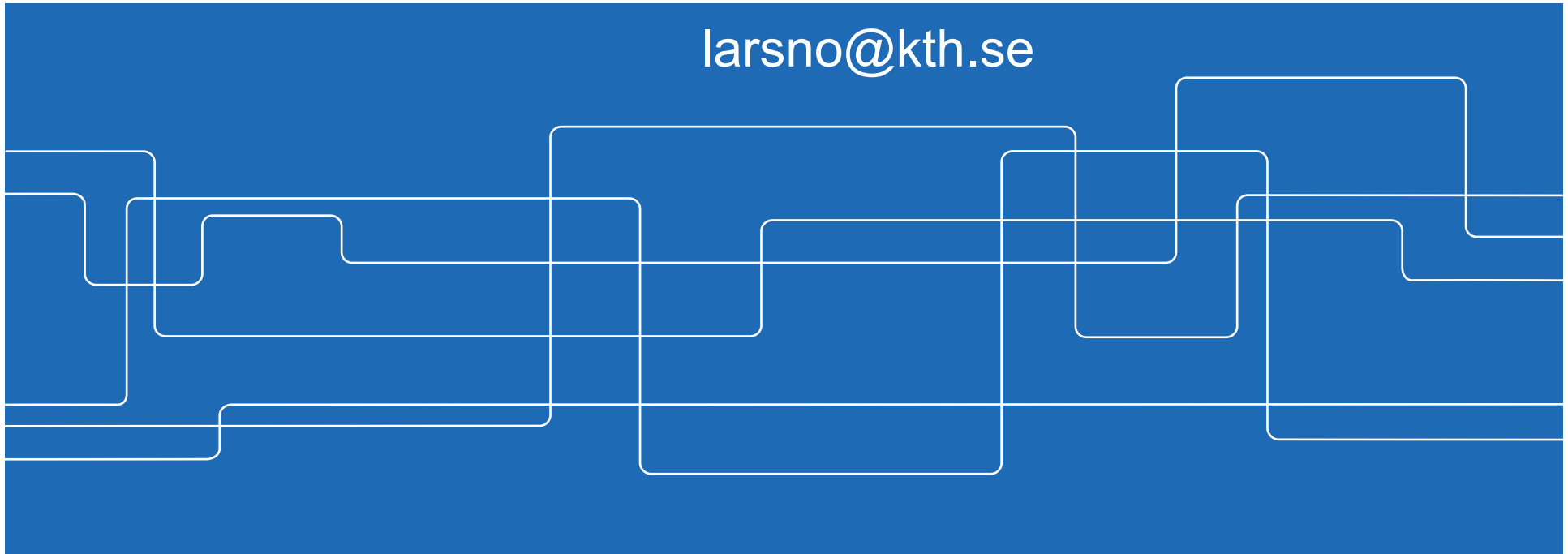




Java and XML parsing

EH2745 Lecture #8
Spring 2015

larsno@kth.se





Lecture Outline

- Quick Review
- The XML "language"
- Parsing Files in Java



Quick Review

We have in the first set of Lectures covered the basics of the Java Language. Next is to make something useful with it.

Like learning a new human language requires practice, so does programming



I/O and Files in Java

My Java Program

JVM (Packages)

OS (Linux, OSX, Win..)

HW (CPU, RAM, HDD, I/O)

Accessing I/O is built on the `InputStream` and `FileSteam`

For keyboard input, this comes from `System.in` to which the `InputStream` is pointed.

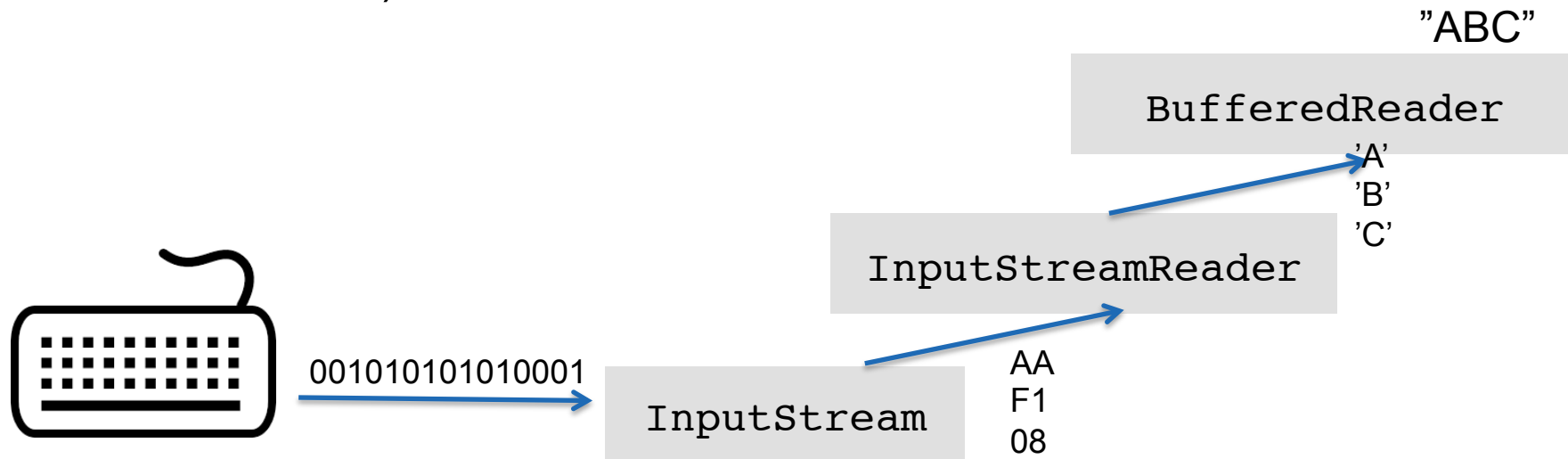
`InputStream` is enhanced in `InputStreamreader` which maps the read Bytes to Characters

For file input, the `FileSteam` is handed a filename



Enhanced reading

Both `InputStreamReader` and `FileReader` can be used by a `BufferedReader`, which reads sets of characters (e.g. a line)



```
FileReader fr = new FileReader("readme.txt");  
BufferedReader br = new BufferedReader(fr);
```



Parsing

Parsing involves analysing a string of characters (or words) and breaking them down into components.

Like if a person reads this long paragraph of text which actually has several words in it but since it is written without any separators it is difficult to see these words fortunately a person is reasonably intelligent and can read this sentence anyway a computer cannot do so however since it is not intelligent it needs some way of telling it which characters should be grouped into groups (words) that can be stored in its memory and used in the program



Parsing - How to convert read lines to data the program can work with

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa

Comma Separate Values (CSV) is a straightforward way to store the data in a file

The program(mer) needs to know the structure of the file.

```
String SplitBy = ",";
List<Flower> flowerList = new ArrayList<Flower>();
br = new BufferedReader(new FileReader(dataFile))
while ((line = br.readLine()) != null) {
    String[] flower = line.split(SplitBy);
    double[] param = {0.0,0.0,0.0,0.0};
    for (int j = 0; j < 4; j++) {
        param[j] = Double.parseDouble(flower[j]);
    }
    flowerList.add(new Flower(param,flower[4]));
}
```



Is the file format important?

5.1,3.5,1.4,0.2,Iris-setosa

4.9,3.0,1.4,0.2,Iris-setosa

4.7,3.2,1.3,0.2,Iris-setosa

4.6,3.1,1.5,0.2,Iris-setosa

5.0,3.6,1.4,0.2,Iris-setosa

5.4,3.9,1.7,0.4,Iris-setosa

4.6,3.4,1.4,0.3,Iris-setosa



Is the file format important?

```
<Flower>
  <Sepal Length>5.0</Sepal Length>
  <Sepal Width>3.6</Sepal Width>
  <Petal Length>0.8</PetalLength>
  <Petal Width>0.2</Petal Width>
  <Species>Iris-Setosa</Species>
</Flower>
<Flower>
  <Sepal Length>5.0</Sepal Length>
  <Sepal Width>3.6</Sepal Width>
  <Petal Length>0.8</PetalLength>
  <Petal Width>0.2</Petal Width>
  <Species>Iris-Setosa</Species>
</Flower>
<Flower>
  <Sepal Length>5.0</Sepal Length>
  <Sepal Width>3.6</Sepal Width>
  <Petal Length>0.8</PetalLength>
  <Petal Width>0.2</Petal Width>
  <Species>Iris-Setosa</Species>
</Flower>
```



Which is the better choice?

- **Comma Separated Values**
 - Pros
 - Little extra data has to be sent (only the commas)
 - Cons:
 - Data has to arrive in the right order
- **Using Tags(XML)**
 - Pros:
 - Flexible format (data can arrive out of order)
 - Thanks to tags we can search for data
 - People can read it!!!
 - Cons:
 - Verbose – a lot of overhead!



Overhead and filesize

What is the overhead in these two examples?

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa

```
<Flower>  
  <Sepal Length>5.0</Sepal Length>  
  <Sepal Width>3.6</Sepal Width>  
  <Petal Length>0.8</PetalLength>  
  <Petal Width>0.2</Petal Width>  
  <Species>Iris-Setosa</Species>  
</Flower>  
<Flower>  
  <Sepal Length>5.0</Sepal Length>  
  <Sepal Width>3.6</Sepal Width>  
  <Petal Length>0.8</PetalLength>  
  <Petal Width>0.2</Petal Width>  
  <Species>Iris-Setosa</Species>  
</Flower>  
<Flower>  
  <Sepal Length>5.0</Sepal Length>  
  <Sepal Width>3.6</Sepal Width>  
  <Petal Length>0.8</PetalLength>  
  <Petal Width>0.2</Petal Width>  
  <Species>Iris-Setosa</Species>  
</Flower>
```



Lecture Outline

- Quick Review
- The XML "language"
- Parsing Files in Java



XML – eXtensible Markup Language

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<note>
```

```
  <to>Tove</to>
```

```
  <from>Jani</from>
```

```
  <heading>Reminder</heading>
```

```
  <body>Don't forget me this weekend!</body>
```

```
</note>
```



XML is (M)a(ny) standard developed by W3C

- XML Core Working Group:
 - XML 1.0 (Feb 1998), 1.1 (candidate for recommendation)
 - XML Namespaces (Jan 1999)
 - XML Inclusion (candidate for recommendation)
- XSLT Working Group:
 - XSL Transformations 1.0 (Nov 1999), 2.0 planned
 - XPath 1.0 (Nov 1999), 2.0 planned
 - eXtensible Stylesheet Language XSL(-FO) 1.0 (Oct 2001)
- XML Linking Working Group:
 - XLink 1.0 (Jun 2001)
 - XPointer 1.0 (March 2003, 3 substandards)
- XQuery 1.0 (Nov 2002) plus many substandards
- XMLSchema 1.0 (May 2001)



A Simple XML Document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</
abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the
universal...
    </section>
  </text>
</article>
```



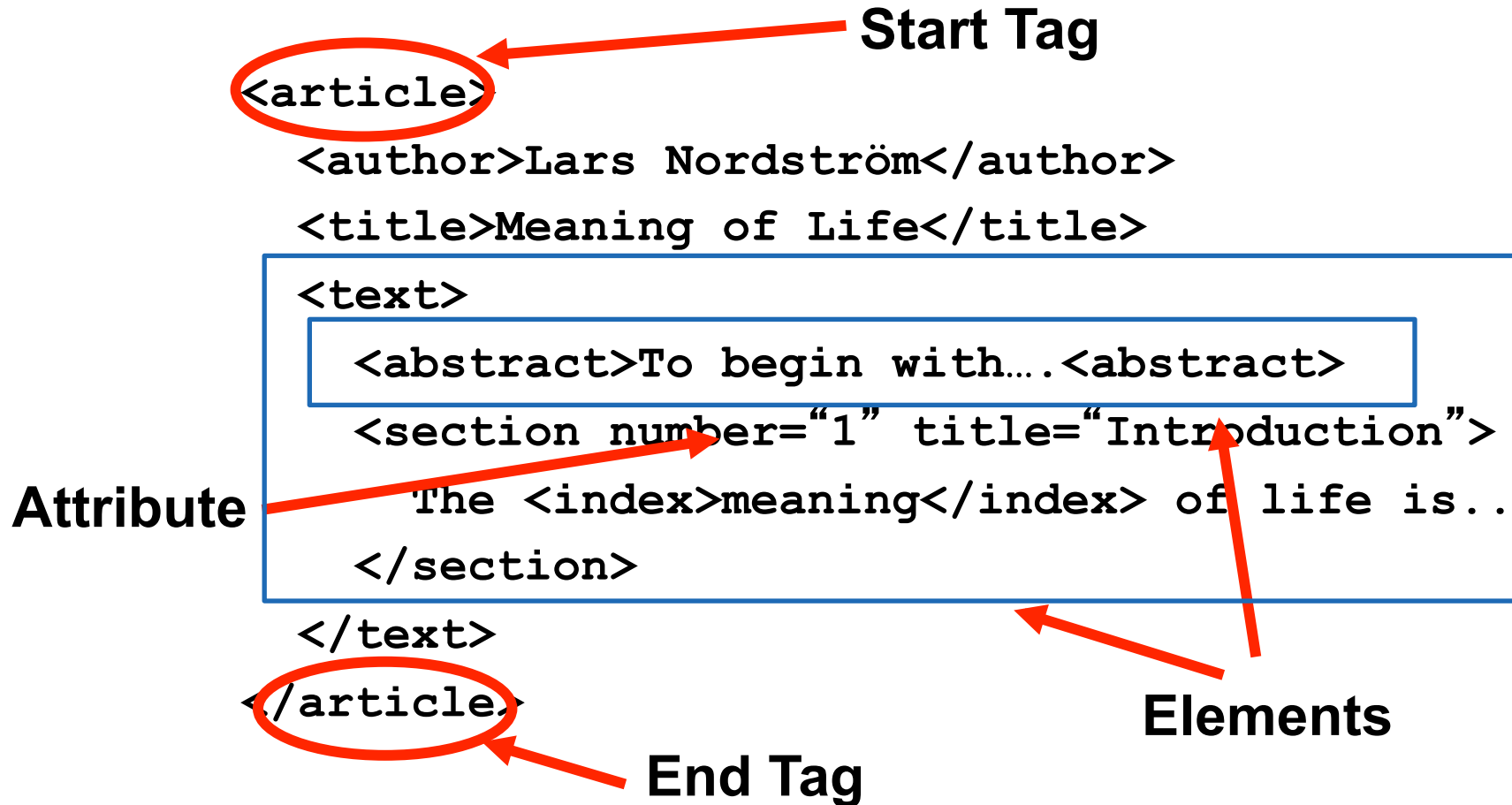
A Simple XML Document

```
<article>
  <author>Ears Nordström</author>
  <title>The Meaning of Life</title>
  <text>
    <abstract>To begin with</abstract>
    <section number="1" title="Introduction">
      The <index>meaning</index> of life is..
    </section>
  </text>
</article>
```

Freely definable tags



A Simple XML Document





Elements in XML Documents

- (Freely definable) **tags**: `article`, `title`, `author`
- with start tag: `<article>` etc.
- and end tag: `</article>` etc.
- **Elements**: `<article> ... </article>`
- Elements have a **name** (`article`) and a **content** (...)
- Elements may be nested.
- Elements may be empty: `<this_is_empty/>`
- Element content is typically parsed character data (PCDATA), i.e., strings with special characters, and/or nested elements



Elements vs. Attributes

- Elements may have **attributes** (in the start tag) that have a **name** and a **value**, e.g. `<section number="1">`.
- What is the difference between elements and attributes?
- Only one attribute with a given name per element (but an arbitrary number of subelements)
- Attributes have no structure, simply strings (while elements can have subelements)
- *As a rule of thumb:*

Content into elements

Metadata into attributes

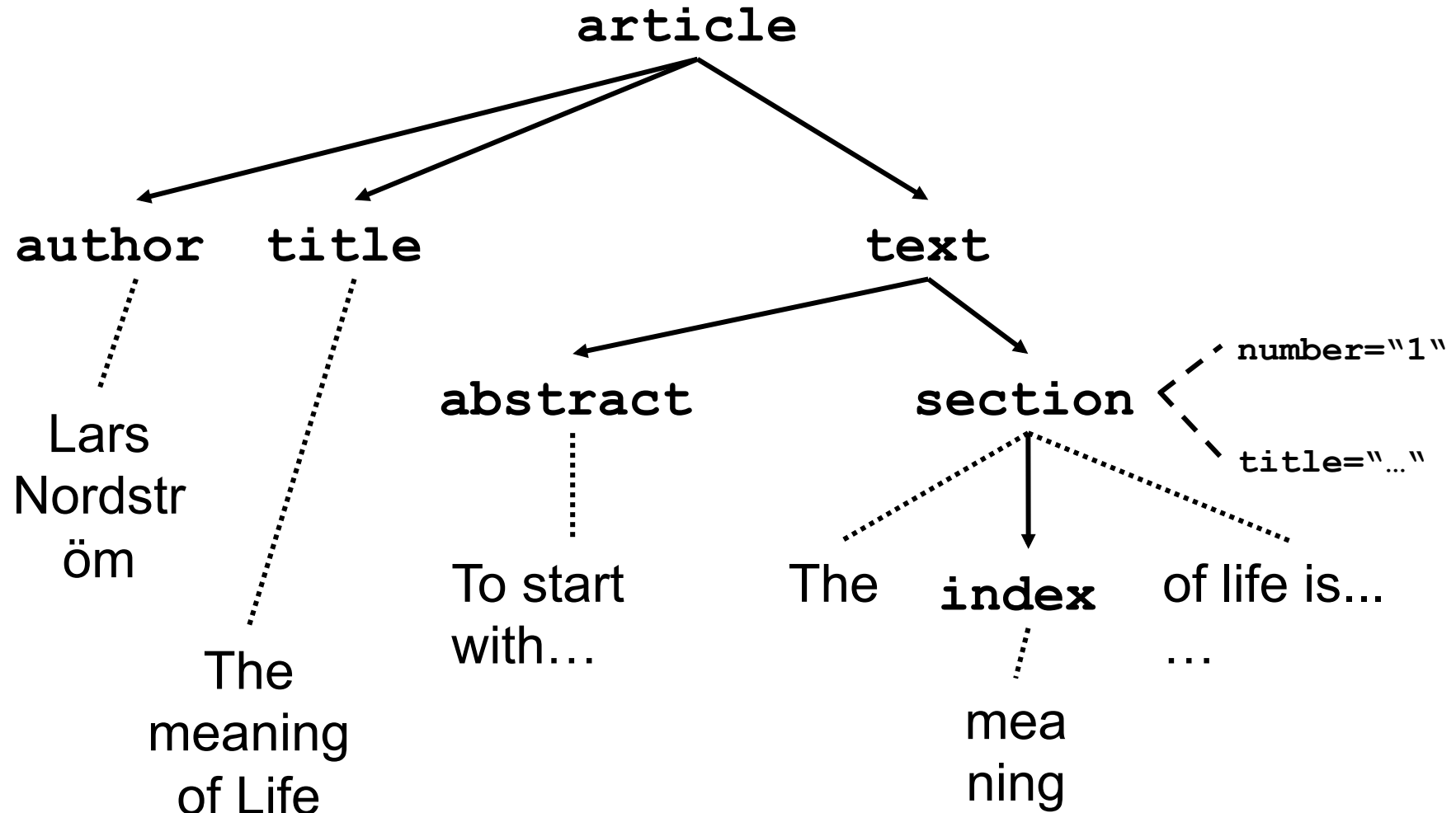
Example:

```
<person born="1912-06-23" died="1954-06-07">
```

```
Alan Turing</person> proved that...
```



XML Documents as Ordered Trees





Well-Formed XML Documents

- A **well-formed** document must adhere to, among others, the following rules:
- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.

Only well-formed documents can be processed by XML parsers.

- All elements may not have two attributes with the same name.
- Comments and processing instructions may not appear inside tags.
- No unescaped < or & signs may occur inside character data.



Namespaces

```
<library>  
  <description>Library of the CS Department</  
  description>  
  <book bid="HandMS2000">  
    <title>Principles of Data Mining</title>  
    <description>  
      Short introduction to <em>data mining</  
      em>, useful  
      for the IRDM course
```

Semantics of the `description` element is ambiguous
Content may be defined differently
Renaming may be impossible (standards!)

⇒ Disambiguation of separate XML applications
using unique prefixes



Namespace Syntax

`<db:book xmlns:db="http://www-dbs/dbs">`

Prefix as abbreviation
of URI

Unique URI to identify the
namespace

Signal that namespace
definition happens



Namespace Example

```
<dbs:book xmlns:dbs="http://www-dbs/dbs">
  <dbs:description> ... </dbs:description>
  <dbs:text>
    <dbs:formula>
      <mathml:math xmlns:mathml="http://
www.w3.org/1998/Math/MathML">
        ...
      </mathml:math>
    </dbs:formula>
  </dbs:text>
</dbs:book>
```




Default Namespace

Default namespace may be set for an element and its content (but *not* its attributes):

```
<book xmlns="http://www-dbs/dbs">  
  <description>...</description>  
</book>
```

Can be overridden in the elements by specifying the namespace there (using prefix or default namespace)



Specifying XML files

How do you specify what the tags should be in an XML file?

DTD – was the initial idea - Document Type definition

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note      (to,from,heading,body)>
  <!ELEMENT to        (#PCDATA)>
  <!ELEMENT from      (#PCDATA)>
  <!ELEMENT heading   (#PCDATA)>
  <!ELEMENT body      (#PCDATA)>
]>
```

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this
weekend!</body>
</note>
```



XML Schema

Why not use XML to specify the format of the XML file?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xlm:lang="en">
      <xsd:element name="bookstore" type="bookstoreType"/>
      <xsd:complexType name="bookstoreType">
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="topic" type="topicType"
minOccurs="1"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:documentation>
  </xsd:annotation>
```



Lecture Outline

- Quick Review
- The XML "language"
- Parsing Files in Java



Parsing XML

We want to read XML files and store them in the working memory so that we can use and manipulate the data.

Possible strategies

- Parse by hand (write from scratch)
- Parse into generic tree structure
- Parse as sequence of events



Parsing into generic tree structure

Advantages

- Industry-wide, language neutral standard exists called DOM (Document Object Model)
- Learning DOM for one language makes it easy to learn for any other
- Have to write much less code to get XML to something you want to manipulate in your program

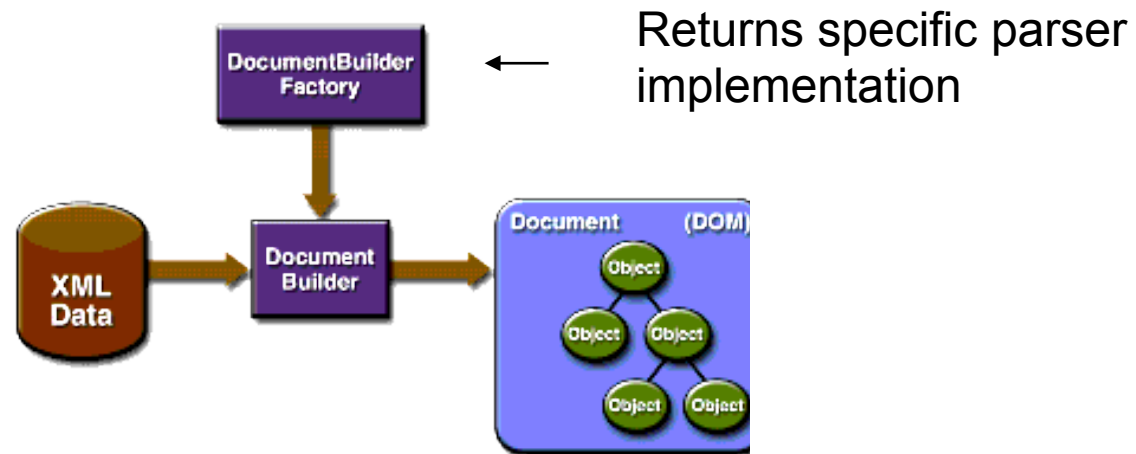
Disadvantages

- Non-intuitive API, doesn't take full advantage of Java
- Still quite a bit of work



DOM abstraction layer in Java -- architecture

Emphasis is on allowing vendors to supply their own DOM Implementation without requiring change to source code



Source: org.w3d.dom.Document



Sample Code

A factory instance
is the parser implementation.

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

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

From the factory one obtains
an instance of the parser

```
Document doc = builder.parse(xmlFile);
```

doc is the root of
the document tree

The method `parse` in the `builder` (which has been created by the factory (by calling the method `newDocumentBuilder` in factory creates the object `doc` by reading the `xmlfile`, this `doc` is of class `Document`



Document object

Once a Document object is obtained, rich API to manipulate.

First call is usually

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

This gets the root element of the Document as an instance of the *Element* class

Note that `Element` subclasses `Node` and has methods `getType()`, `getName()`, and `getValue()`, and `getChildNodes()`



Types of Nodes

Note that there are many types of Nodes (ie subclasses of Node):

`Attr, CDATASection, Comment, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction, Text`

Each of these has a special and non-obvious associated type, value, and name.

Standards are language-neutral and are specified on chart on following slide



Node	nodeName()	nodeValue()	Attributes	nodeType()
Attr	Attr name	Value of attribute	null	2
CDATASection	#cdata-section	CDATA content	null	4
Comment	#comment	Comment content	null	8
Document	#document	Null	null	9
DocumentFragment	#document-fragment	null	null	11
DocumentType	Doc type name	null	null	10
Element	Tag name	null	NamedNodeMap	1
Entity	Entity name	null	null	6
EntityReference	Name entity referenced	null	null	5
Notation	Notation name	null	null	1
ProcessingInstruction	target	Entire string	null	7
Text	#text	Actual text	null	3



Lecture Outline

- Quick Review
- The XML "language"
- Parsing Files in Java