UNIVERSITY OF MANNHEIM

Web Data Integration

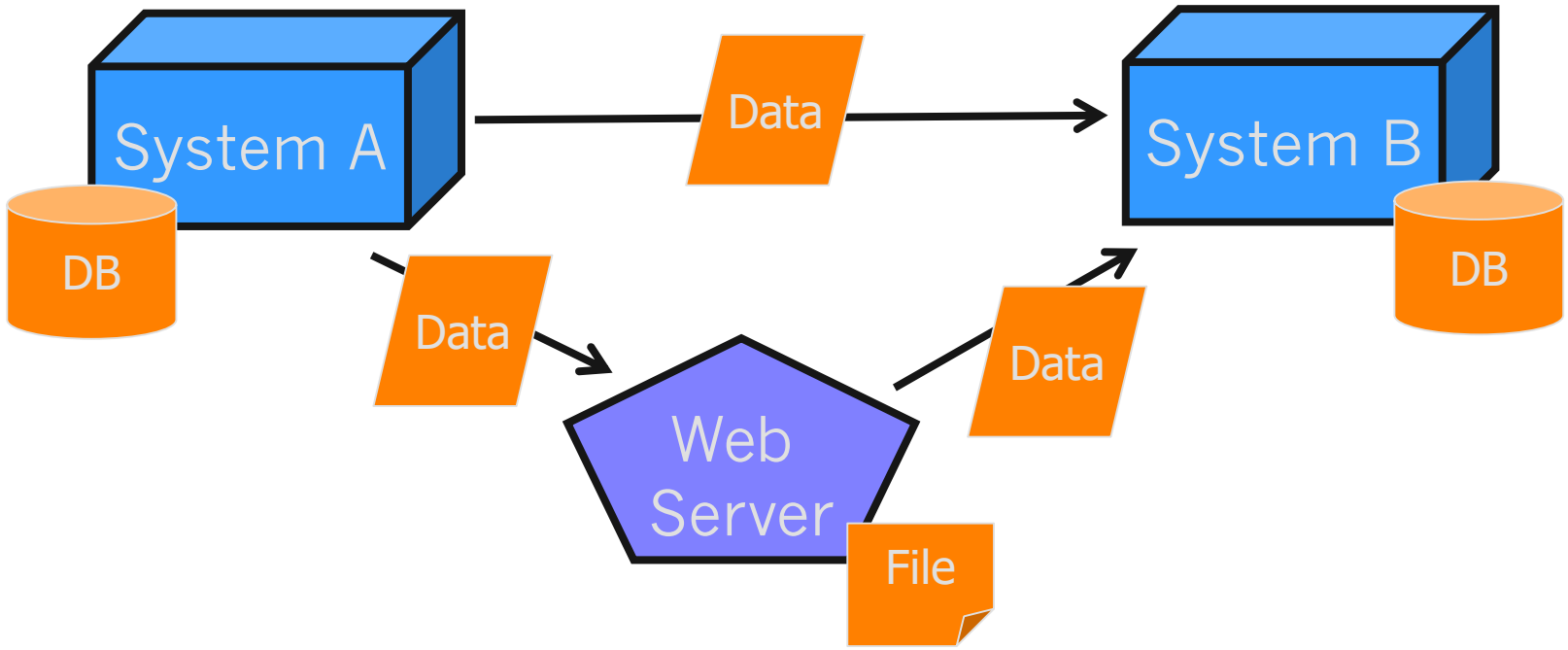
Data Exchange Formats

- Part 1 -

Data Exchange

Data Exchange: Transfer of data from one system to another.

Data Exchange Format: Format used to represent (encode) the transferred data.



Web Data

Web Data is heterogeneous with respect to the employed

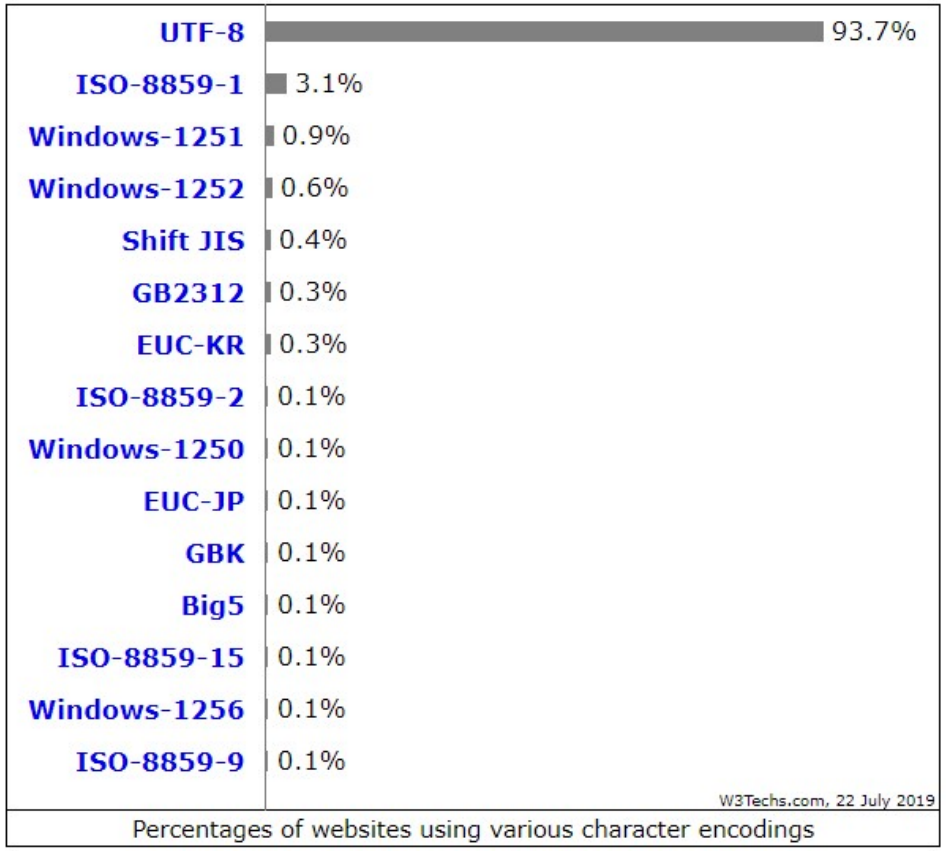
1. **Data Exchange Format (Technical Heterogeneity)**
2. **Character Encoding (Syntactical Heterogeneity)**

Outline

1. Data Exchange Formats - Part I
    1. Character Encoding
    2. Comma Separated Values (CSV)
        1. Variations
        2. CSV in Java
    3. Extensible Markup Language (XML)
        1. Basic Syntax
        2. DTDs
        3. Namespaces
        4. XPath
        5. XSLT
        6. XML in Java
2. Data Exchange Formats - Part II
    1. JavaScript Object Notation (JSON)
    2. Resource Description Framework (RDF)

Character Encoding

- Every character is represented as a bit sequence, e.g. “A” = 0100 0001
- Character encoding: mapping of “real” characters to bit sequences
- A common problem in data integration:



http://w3techs.com/technologies/overview/character_encoding/all

http://geekandpoke.typepad.com/geekandpoke/2011/08/coders-love-unicode.html

Character Encoding: ASCII, ISO 8859

- ASCII („American Standard Code for Information Interchange“)
  ISO 646 (1963), 127 characters (= 7 bits), 95 printable:

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

- Extension to 8 Bits: ISO 8859-1 to -16 (1998)
  - covers characters of European languages
  - well-known: 8859-1 (Latin-1)
  - including: Ä, Ö, Ü, ß, Ç, É, é, …
- But the Web speaks more languages...

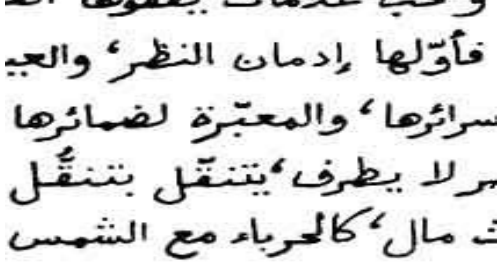
Character Encoding: Unicode

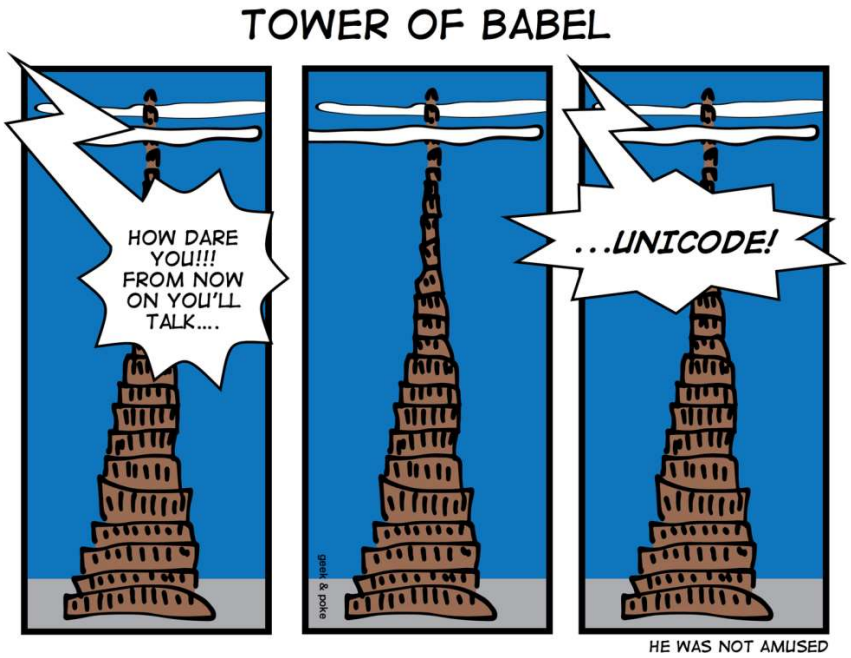
ISO 10646

- first version 1991 (Europe, Middle East, India)
- 17 code pages of 16 bit
- covers even the most exotic languages

我爱中国
国中爱我

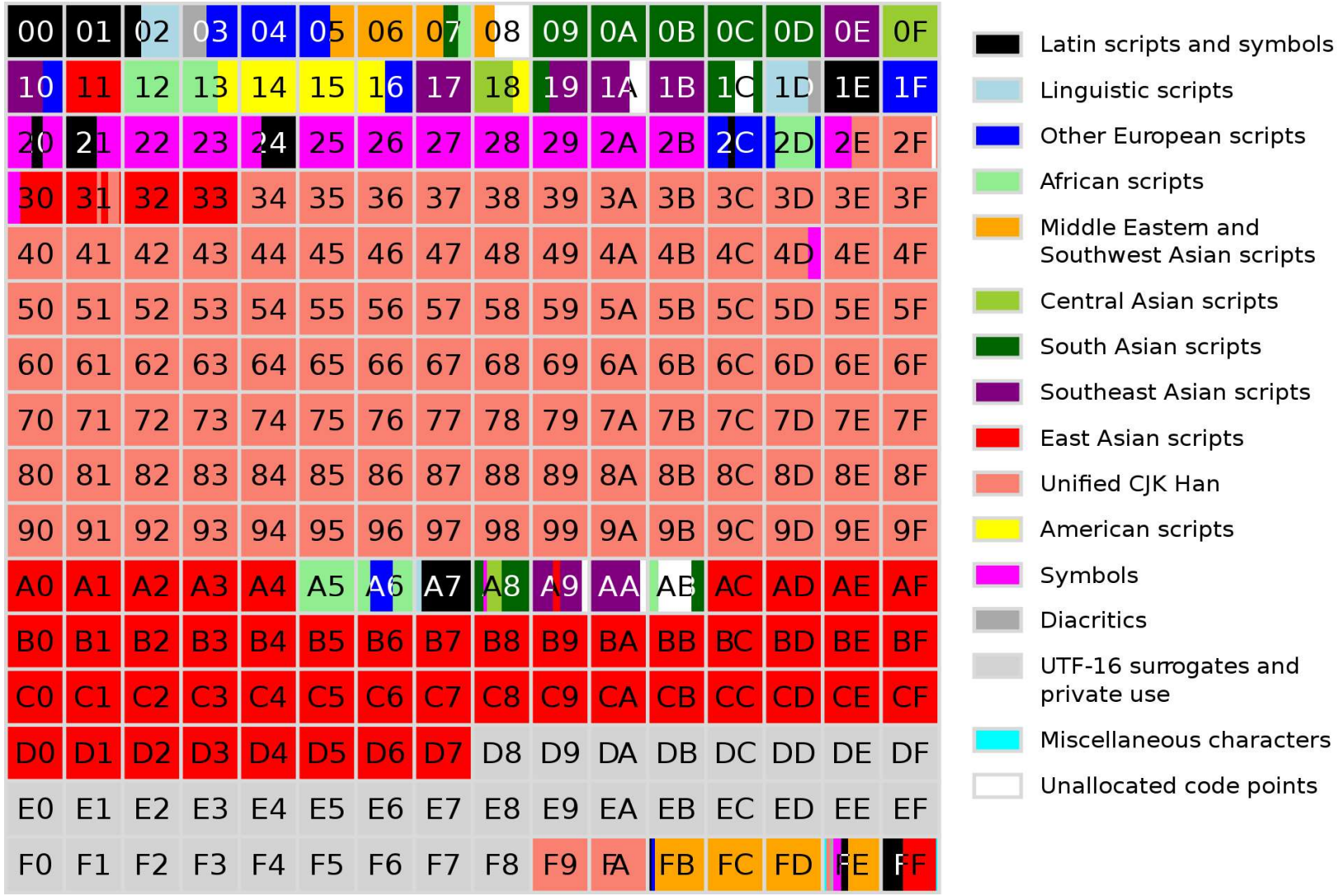
ถ้าผิมขรูน
ตัวอักษรย่างนี้
คุณจะเข้าใจไหมขรับ





http://geek-and-poke.com/geekandpoke/2013/8/29/when-it-all-began

Character Encoding: Unicode



Source: Wikimedia Commons

Character Encoding: UTF-8

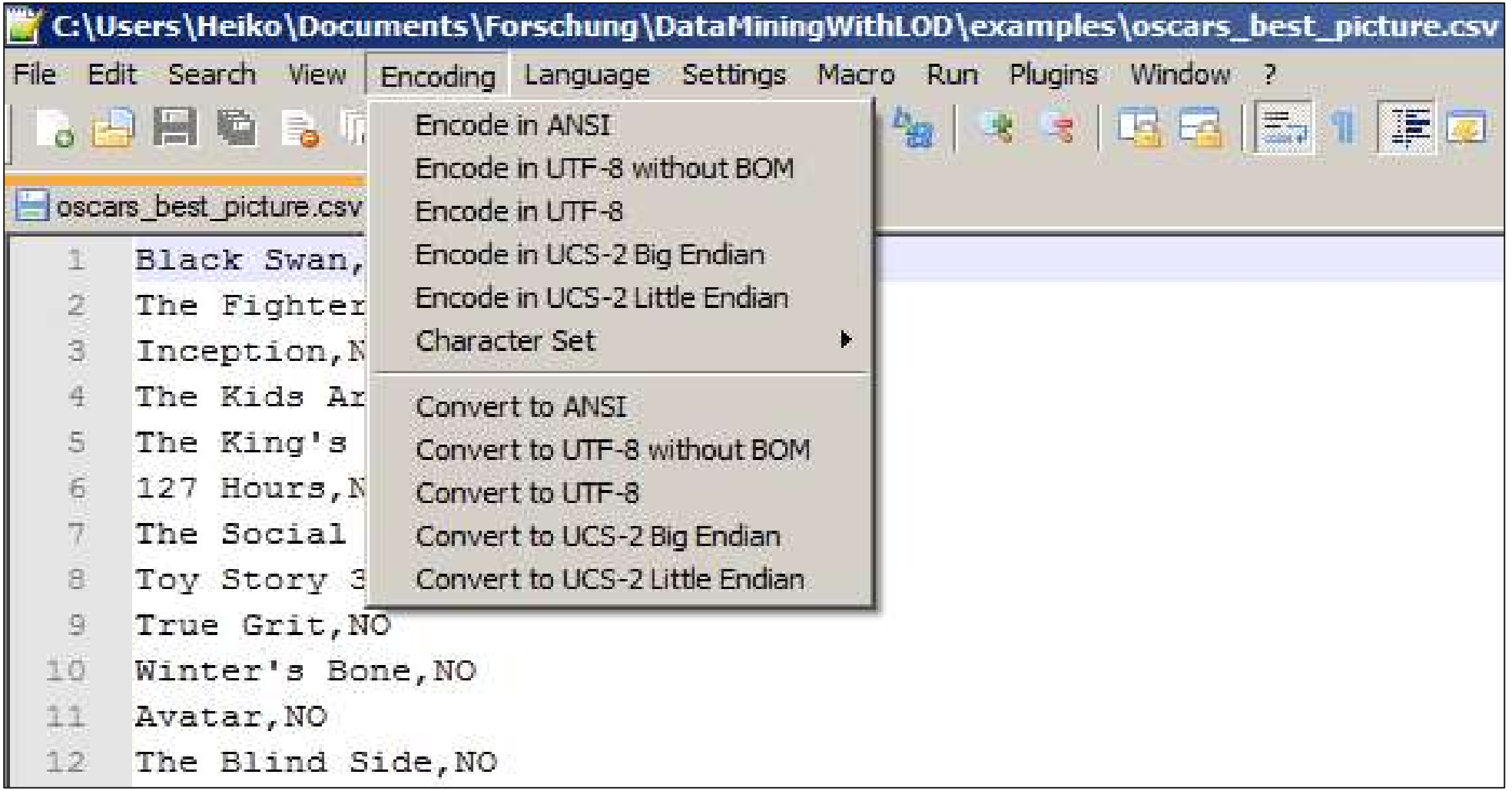
- UTF-8: Variable length encoding for Unicode
- Recommended character encoding for the Web
- Rationale:
  - common characters are encoded using only one byte
  - less common ones are encoded in 2-6 bytes
  - fast transmission of files over the internet!

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Source: Wikipedia

Handling Character Encoding

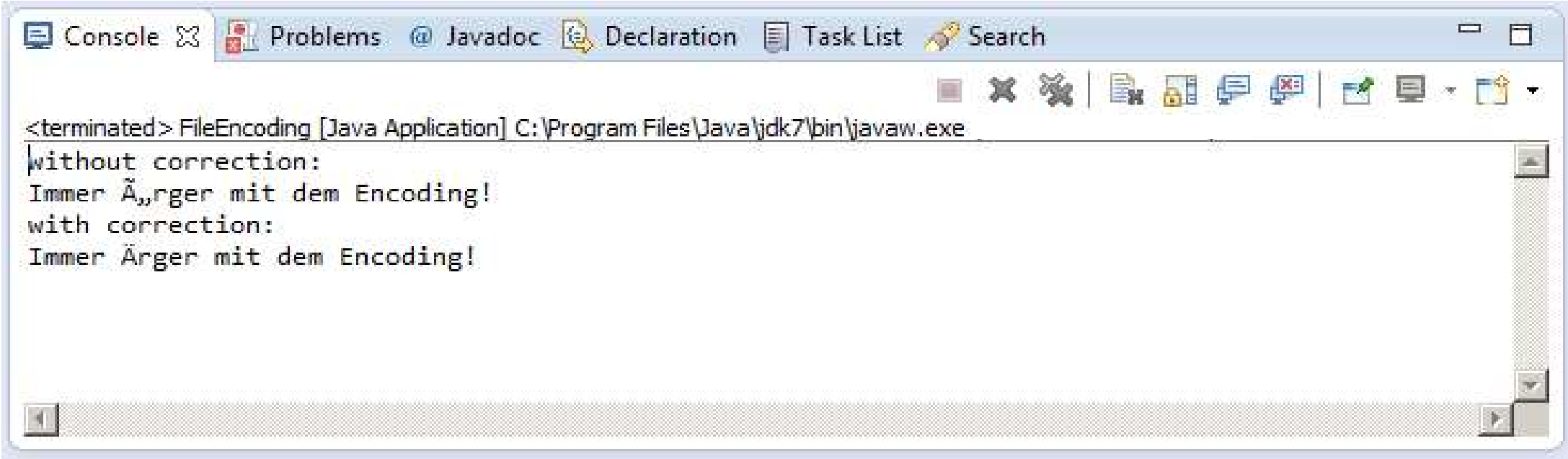
Editors such as **Notepad++** support encoding conversion.



Handling Character Encoding in Java

FileInputStreams allow you to specify the character encoding.

```
BufferedReader BR = new BufferedReader(
   new InputStreamReader(
   new FileInputStream("data/encoding_utf8.txt"),"UTF8"));
while(BR.ready())
   System.out.println(BR.readLine());
```



Handling Character Encoding in XML

CD2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CD SYSTEM "CD.dtd">
<!--  Description of a CD   -->
<CD ArticelNo="2">
      <Artist>Moby</Artist>
      <Album>Play</Album>
      <ReleaseDate>
           03.06.2000
      </ReleaseDate>
      <Label>Mute (EDEL)</Label>
      <Format>CD</Format>
</CD>
```

Encoding is specified in document prolog as Web documents should be self-descriptive.

2. Comma Separated Values (CSV)

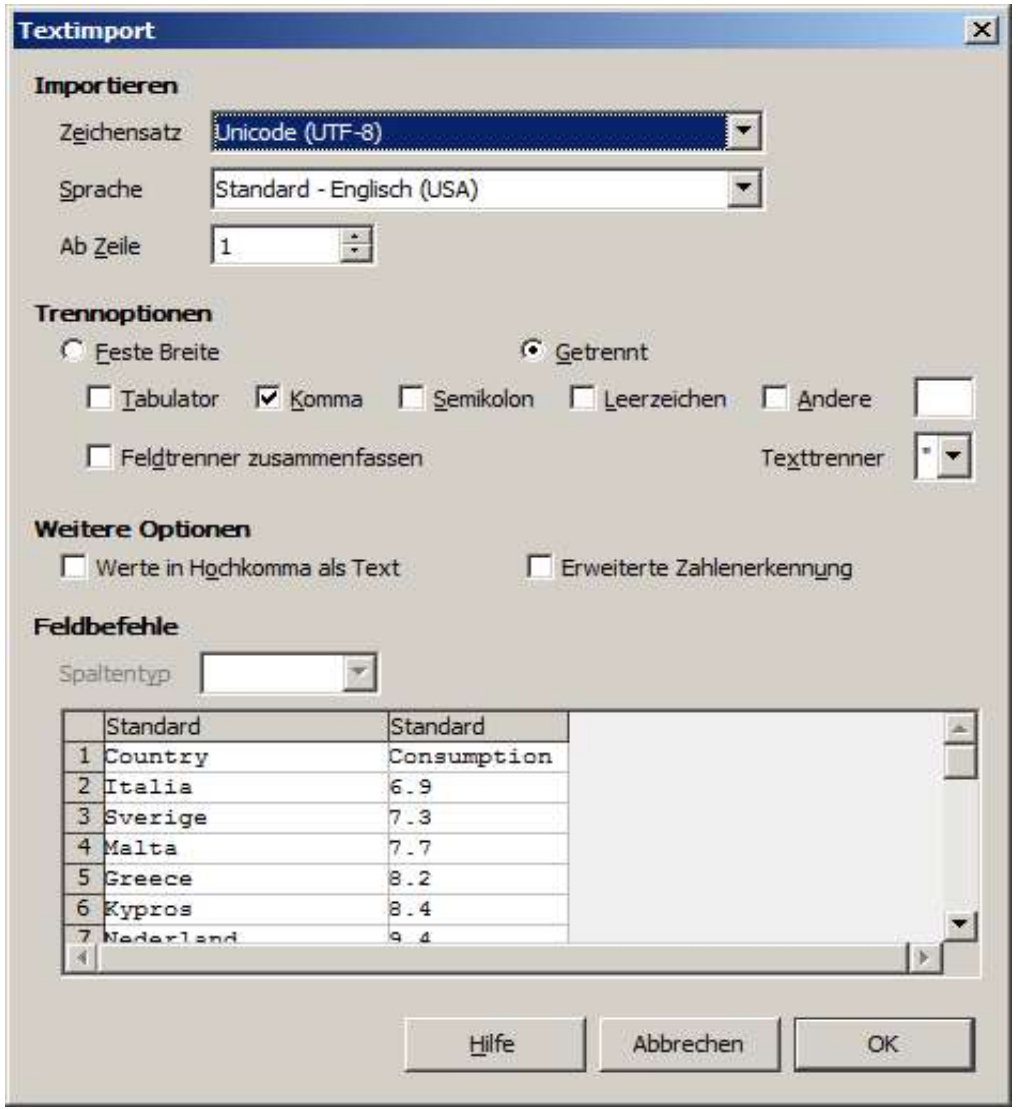
- Data model: Table
  - used for data exported from RDBMs and spreadsheet applications
  - quite widely used on the Web and on public data portals
  - the first line is often used for headers (attribute names)
- Example:

```
firstname,lastname,matriculation,birthday
thomas,meyer,3298742,15.07.1988
lisa,müller,43287342,21.06.1989
```

- Advantage: Data representation with minimal overhead
- Disadvantages
  - restricted to tabular data
  - hard to read for humans when tables get wider
  - different variations, no support for data types

Comma Separated Values (CSV) - Variations

- Field Separators
  - comma, semicolon, tab, …
- Quotation marks
  - for marking strings
- Header included
  - nor not
- Dealing with the variations
  1. configuration
  2. automatic detection
  3. standardized metadata: W3C Tabular Data and Metadata on the Web
     https://www.w3.org/TR/tabular-data-primer/



Processing CSV Files in Java

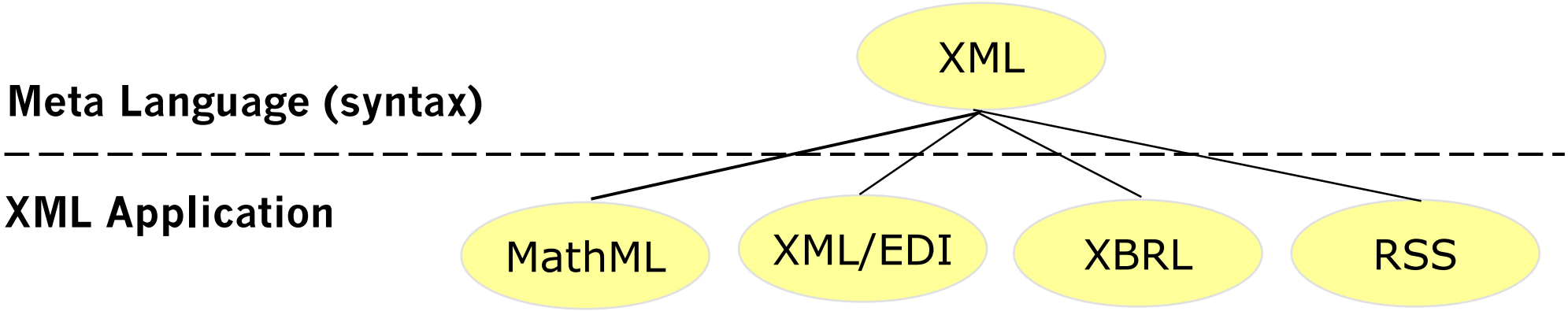
- Apache Commons CSV
- Provides simple API for iterating over CSV files
- http://commons.apache.org/proper/commons-csv/
- Example:

```
Reader in = new FileReader("data/data.csv");
Iterable<CSVRecord> parser = CSVFormat.EXCEL.parse(in);
for (CSVRecord record : parser) {
  if(record.getRecordNumber()>1) {
    String firstname = record.get(0);
    String lastname  = record.get(1);
    ...
  }
}
```

skip header line

3. XML - eXtensible Markup Language

- Standardized by W3C in 1998
- Widely used format for data exchange in the Web and enterprise contexts
- Data model: Tree
- Meta language
  - defines standard syntax
  - allows the definition of specific languages (XML applications)



3.1 XML – Basic Concepts and Syntax

1. Elements
   - Enclosed by pairs of tags:

     <physician> … </physician>
   - Empty elements:

     <young />
2. Attributes

   <physician id="D125436">
3. Hierarchy
   - exactly one root element!

   <physician>

   <address> … </address>

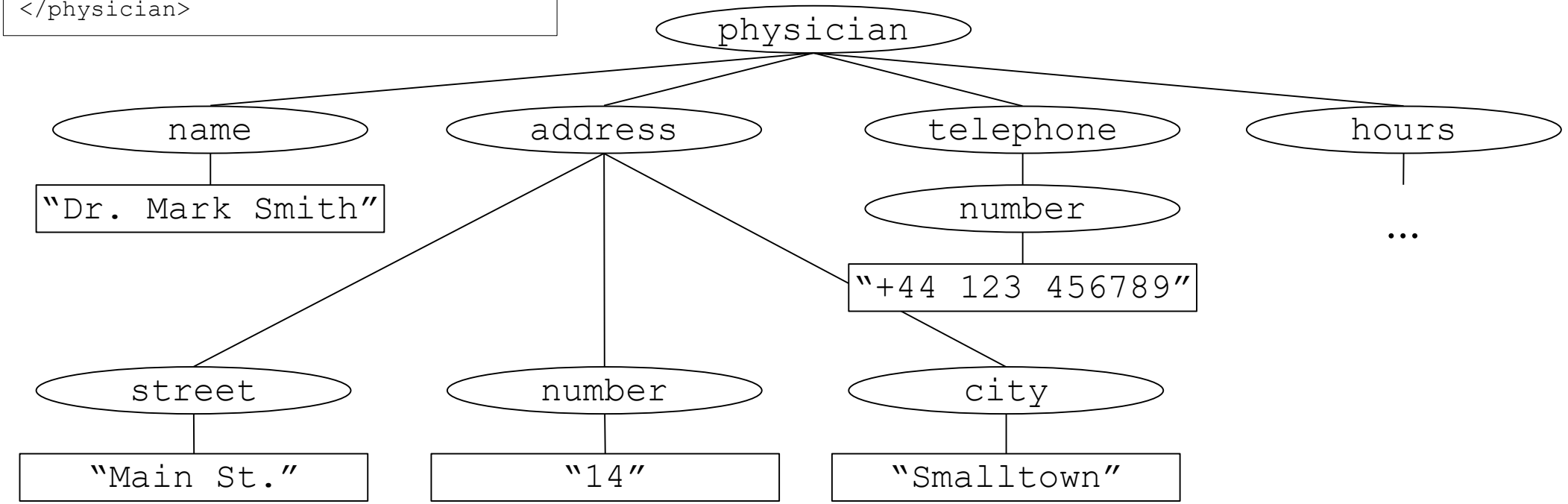
   <telephone> … </ telephone>

   </physician>

```
<physician id="D125436">
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

XML as a Tree

```
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    ...
  </hours>
</physician>
```



HTML versus XML

- HTML: Aimed at displaying information to humans
  - mixes structure, content, and presentation
- XML: Aimed at data exchange
  - separates structure, content, and presentation

```
<html>
  …
  <b>Dr. Mark Smith</b>
  <i>Physician</i>
  Main St. 14
  Smalltown
  …
</html>
```

```
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
</physician>
```

Overall Structure of an XML Document

CD2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CD SYSTEM "CD.dtd">
<!--  Description of a CD  -->
<CD ArticelNo="2">
      <Artist>Moby</Artist>
      <Album>Play</Album>
      <ReleaseDate>
           03.06.2000
      </ReleaseDate>
      <Label>Mute (EDEL)</Label>
      <Format>CD</Format>
</CD>
```

Prolog (required)

Document Type Definition (optional)

Comments (optional)

Root Element (required)

Additional Elements (optional)

Well-formed XML Documents

Document that complies to the syntax requirements of XML

1. Closing tag for each opening tag
2. Proper nesting of tags
3. Only one attribute with a specific name, …

Well-formed

```
<physician id="D1254" >
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

Not well-formed

```
<physician id="D1254" id="US43759">
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  <telephone>
    <number>+44 123 456789</number>
  </address>
  </telephone>
  <hours>
    <monday>9-11 am
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

Sometimes, we need more than trees …

```
<student>
  <name>Stefanie Müller</name>
  <course>
    <title>Web Data Integration</title>
  </course>
  <course>
    ...
  </course>
</student>
<student>
   <name>Franz Maier</name>
   <course>
    <title>Web Data Integration</title>
  </course>
  ...
```

If we organize the XML by students, we have to replicate courses

```
<course>
  <title>Web Data Integration</title>
  <student>
    <name>Stefanie Müller</name>
  </student>
  <student>
     <name>Franz Maier</name>
  </student>
  ...
</course>
<course>
   <title>Data Mining</title>
   <student>
    <name>Stefanie Müller</name>
   </student>
   ...
```

If we organize the XML by courses, we have to replicate students

XML References

- Trees are limited when it comes to n:m relations
- Problem: data duplication
  - consistency
  - storage
  - transmission volume
- Solution: IDs and references

```
<student id="stud01">
  <name>Stefanie Müller</name>
</student>
<student id="stud02">
  <name>Franz Maier</name>
</student>
<course>
  <title>Data Integration</title>
  <lecturer>
    <name>Christian Bizer</name>
  </lecturer>
  <attendedBy ref="stud01" />
  <attendedBy ref="stud02" />
</course>
```

The XML Standards Family

- XML: Meta language for defining markup languages; provides standard syntax
- DTD: Language for defining the structure of XML documents; XML applications
- XML Schema: More expressive language for defining the structure of XML documents, includes data types
- Namespaces: Mechanism for distinguishing between elements from different schemata
- XPath: Language for selecting parts of an XML document
- XQuery: Query language; more flexible than XPath; similar to SQL
- XSLT: Template language for transforming XML documents; uses XPath
- DOM, SAX: Standardized programming interfaces for accessing XML documents from within different programming languages
- XPointer: XML application for defining hyperlinks between elements in different XML documents; combines URLs and Xpath

3.2 Document Type Definition (DTD)

- Defines valid content structure of an XML document
  - allowed elements, attributes, child elements, optional elements
  - allowed order of elements
- DTDs can be used to validate an XML document from the Web before it is further processed.
- XML documents are called “valid” if they are
  - well-formed (syntactically correct)
  - and suit a DTD
- DTD is part of the W3C XML Specification

Referring from a Document to its DTD

physician.dtd

```
<!ELEMENT physician (
  name,
  address*,
  telephone?,
  fax?,
  hours)>

<!ELEMENT address (
  street,
  number,
  city)>

<!ELEMENT street (#PCDATA)>
<!ELEMENT number (#PCDATA)>

  ...

]>
```

examp.xml

```
<!DOCTYPE physician SYSTEM
  "physician.dtd">

<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

Document Type Definition (DTD)

1. Defining child elements and their order

```
<!ELEMENT address(street,nr,addline*,zip,city,state?) >
```

   - ? marks optional, * marks repeatable elements, + means at least once
   - #PCDATA: Parsed character data that may include further elements.
   - #CDATA: Character data that is not parsed.
   - alternative elements `<!ELEMENT TextIncBold ((#PCDATA | B)*)>`

2. Defining attributes

```
<!ATTLIST person number ID #REQUIRED
                title CDATA #IMPLIED
                supervisor IDREF #IMPLIED>
```

   - #REQUIRED = value necessary
   - #IMPLIED = no value necessary
   - ID and IDREF are used to define references

Example: A Complete DTD

CD.dtd

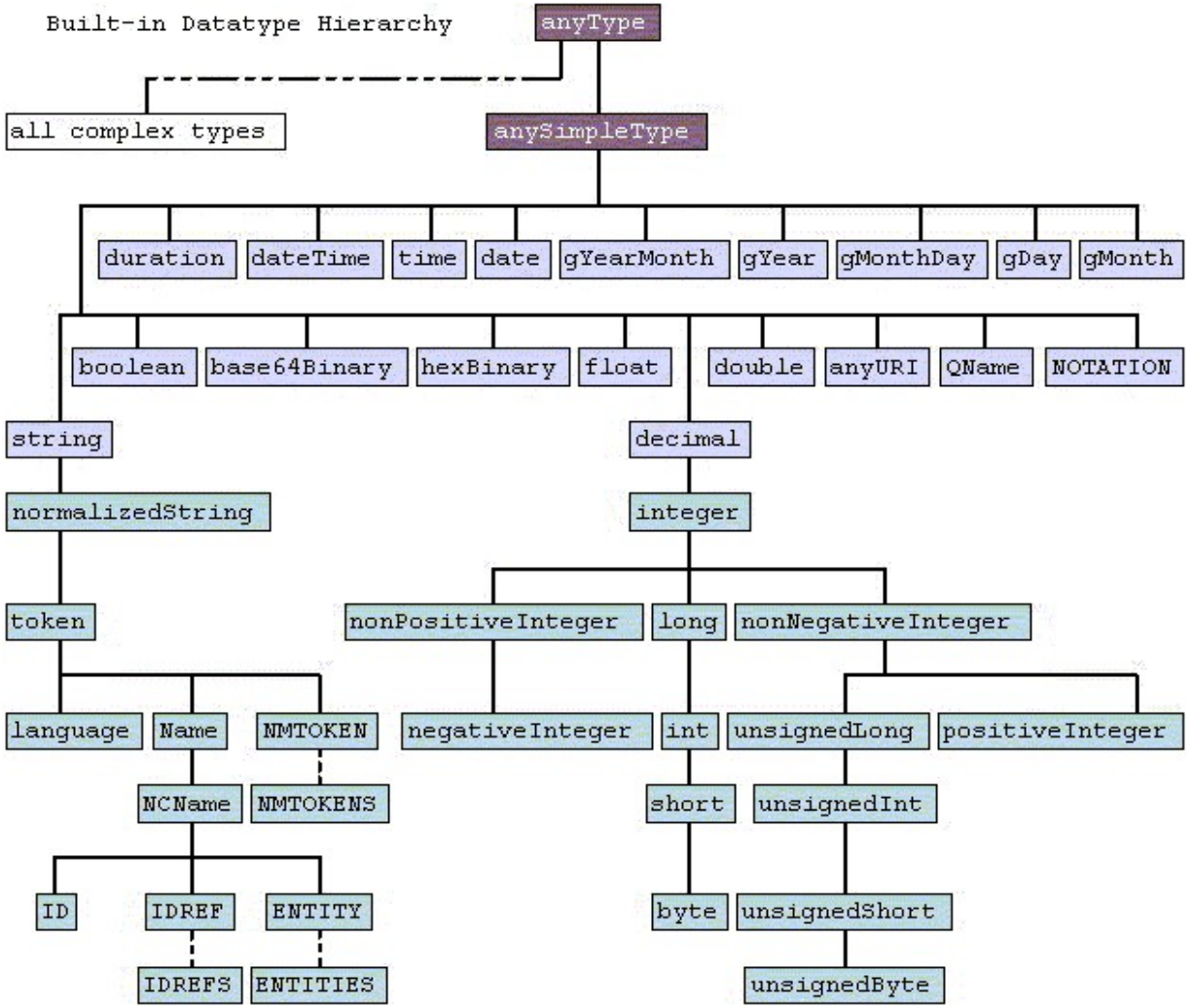
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document Type Definition: CD-Example -->

<!ELEMENT CD (Artist, Album,
              ReleaseDate, Label, Format)>
<!ATTLIST CD ArticleNo CDATA #REQUIRED>

<!ELEMENT ReleaseDate (#PCDATA)>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT Artist (#PCDATA)>
<!ELEMENT Label (#PCDATA)>
<!ELEMENT Album (#PCDATA)>
```

XML Schema

- More flexible than DTDs
  - minimum and maximum number of elements
  - data types (numbers, dates, …)
  - support for namespaces
  - modular schemas are possible
- Standardized by W3C (2004)
- XML Schema documents are XML documents themselves
  - unlike DTDs
  - but more verbose syntax

XML Schema Data Types

- Simple data types are built in
- complex types can be defined by the user
- XML schema data types are also used by RDF



XML Schema Part 2: Datatypes Second Edition
http://www.w3.org/TR/xmlschema-2/

3.3 XML Namespaces

- Problem: Elements with the same name but different meaning (homonyms) may occur in different schemata.
- How can we distinguish such elements if schemata are mixed in the same document?

```
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

XML Namespaces

Mechanism for distinguishing between elements from different schemata by naming them with URIs.

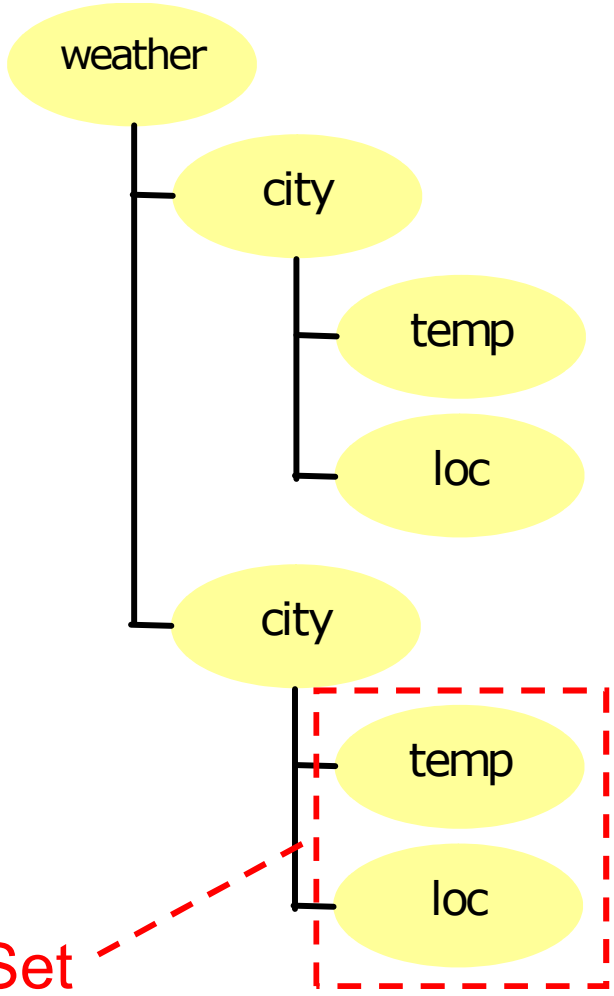
- Shorthand notation with prefix for qualified names (QNames): `prefix:localname`
- Default namespace `xmlns =` and additional namespaces `xmlns:addr =`

```
<physician xmlns = "http://www.med.com/physician/"
           xmlns:addr = "http://www.med.com/addr/">
  <name>Dr. Mark Smith</name>
  <addr:address>
    <addr:street>Main St.</addr:street>
    <addr:number>14</addr:number>
    <addr:city>Smalltown</addr:city>
  </addr:address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

3.4 XPath

Language for selecting sets of nodes from an XML document.

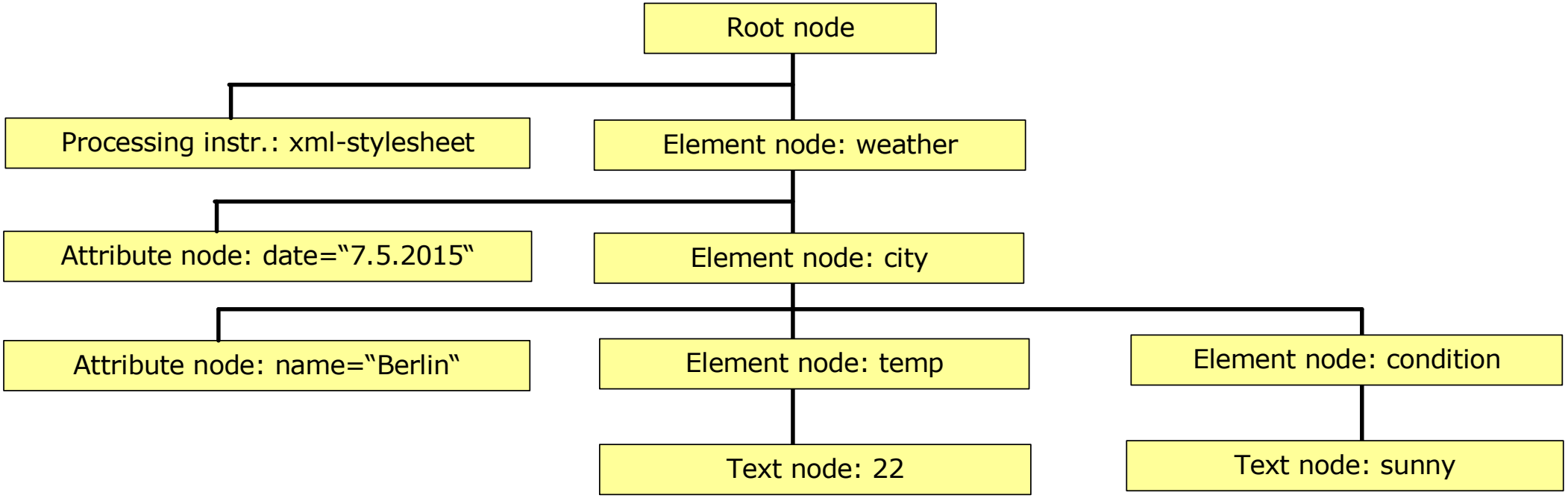
- W3C standard since 1999 (Version 2.0: 2010)
- Used by
  - XSLT
  - XPointer
  - XML Databases
  - Java JAXP API
- Result of a XPath expression: Node Set
- Tutorial: https://www.w3schools.com/xml/xpath_intro.asp



XPath Node Types

Node Type	Explanation
Root node	Abstract root of XML tree Note: This node lies one level above the root XML element and is used to access processing instructions
Processing instruction node	Processing instructions are for instance references to style sheets. All lines that start with <? and end with ?>
Element node	Each element of the document (Start-Tag ... End-Tag)
Attribute node	Each attribute of an element (e.g. date = “5/1/2017“)
Text node	Largest possible connected character sequence. Example: The element <word>Chris</word> contains two text nodes: „C“ and „hris“

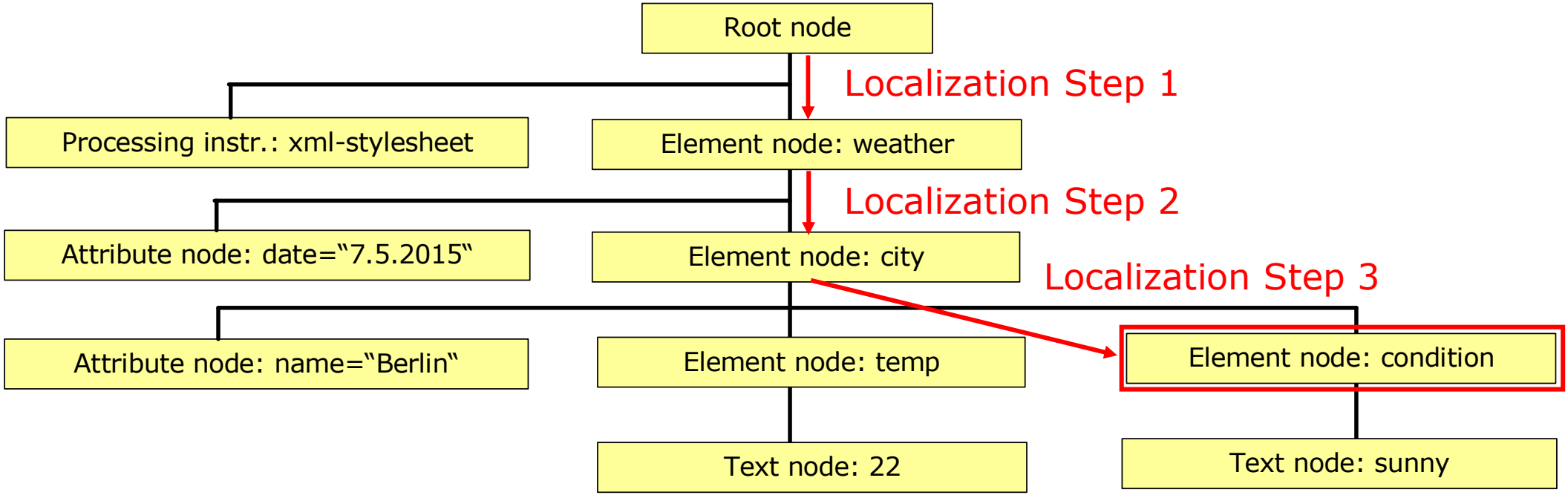
Example: XPath Node Types




```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="weather.css"?>
<weather date="7.5.2015">
    <city name="Berlin">
        <temp>22</temp>
        <condition>sunny</condition>
    </city>
</weather >
```

Selecting Node Sets with XPath

- Nodes are addressed using localization paths.
- Each path consists of a series of localization steps, similar to addressing files in the file system.
- Example: /weather/city/condition

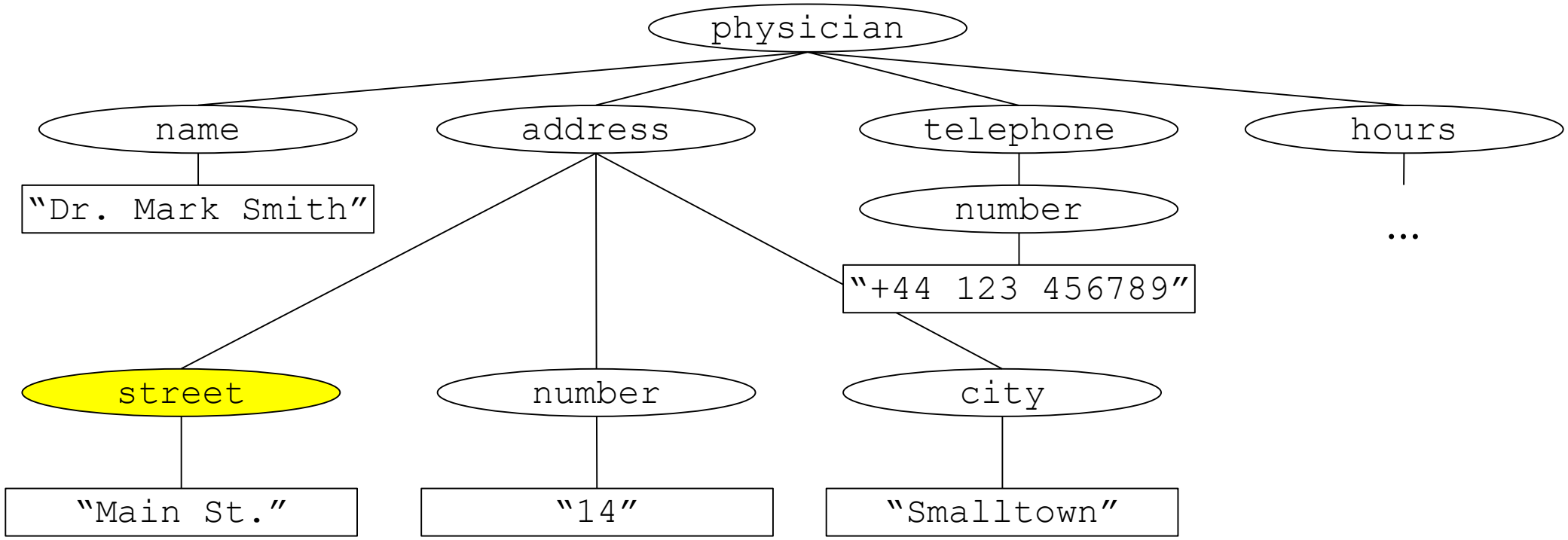


XPath Operators

Syntax	Explanation
weather	Element nodes are addressed using their name
@date	Attribute nodes are addressed using @name (Result: date = “7.5.2015“)
temp/text()	Text nodes are addressed using /text() (Result: 22)
/	Selection of the root node
//temp	Selection of all temp-nodes, no matter on which level(s) of the document (Result: temp)
city/* city/@* city/*[2]	The /* operator selects all child element nodes The /@* operator selects all child attribute nodes (Result1: temp, condition; Result2: name = “Berlin“; Result3: condition)
city/node()	Selection of all child nodes independent of their type. (Result: temp, condition, name = “Berlin“)
//city/..	Using .. it is possible to address the parent elements of an element (Result: weather).
//temp | //condition	The and operator allows several localization paths to be applied in parallel

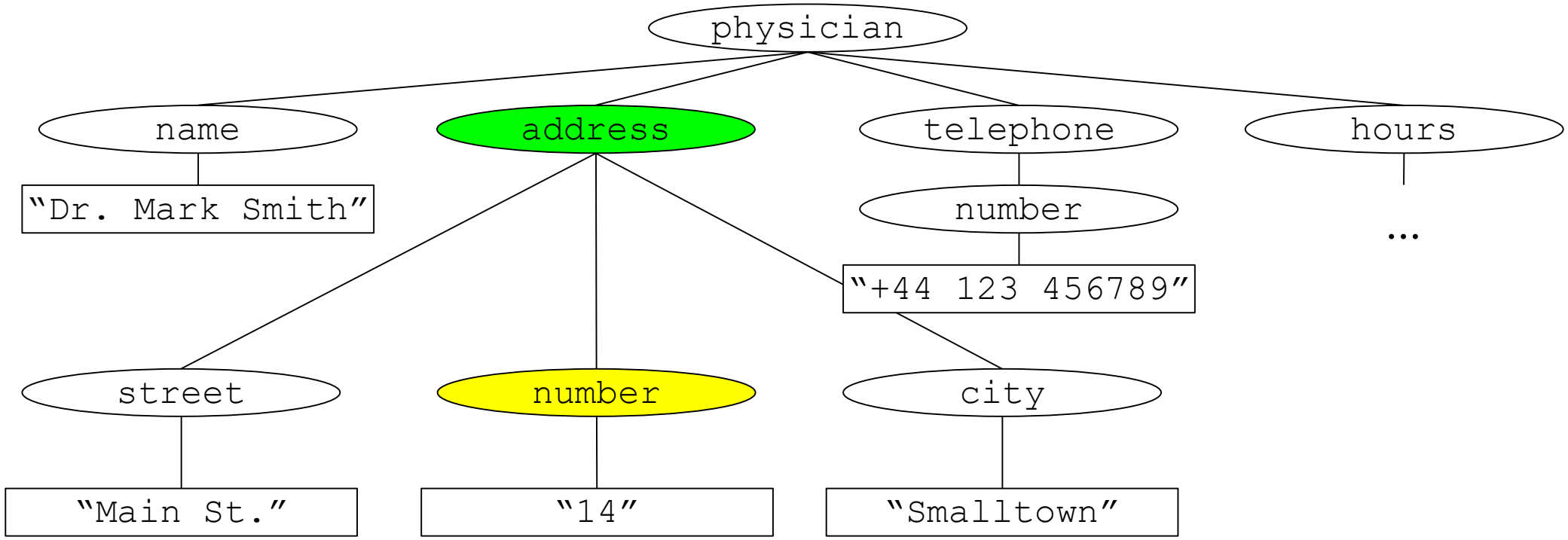
XPath Example 1: Localization starts at Root Node

- Example: `/physician/address/street`
- First / stands for root node



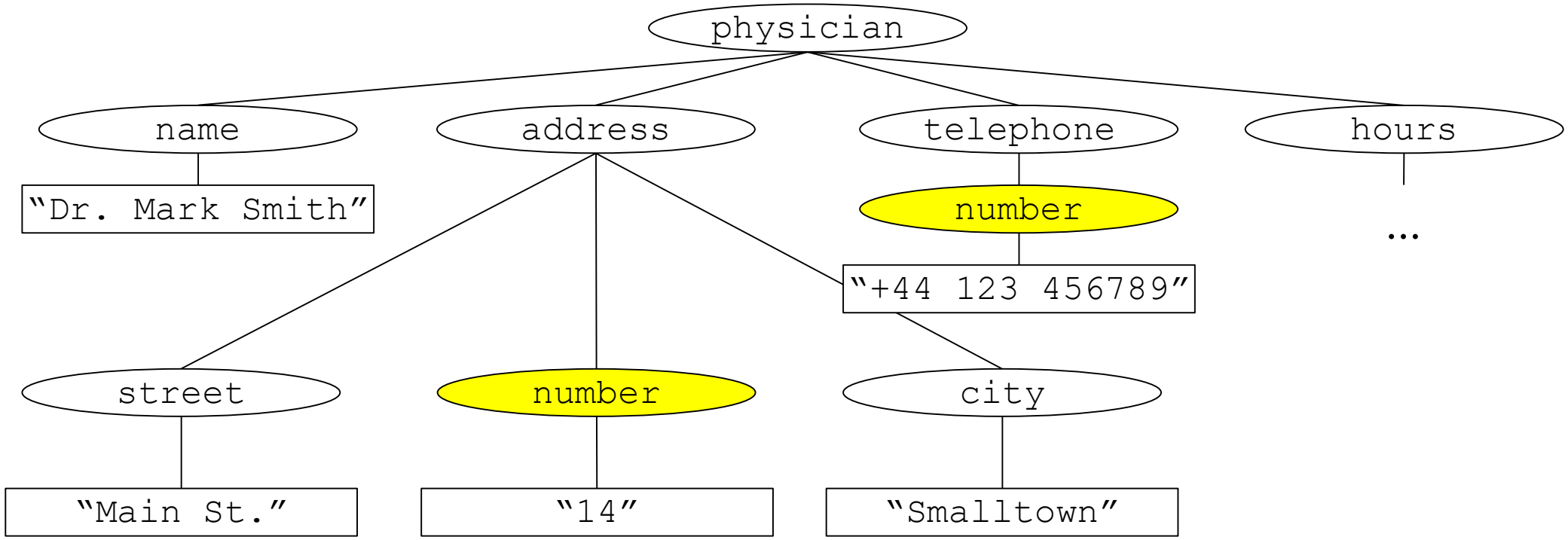
XPath Example 2: Localization starts at Context Node

- Example: `number`
- No / before first element means start at context node (marked green)
- Context node exists for instance in XSLT



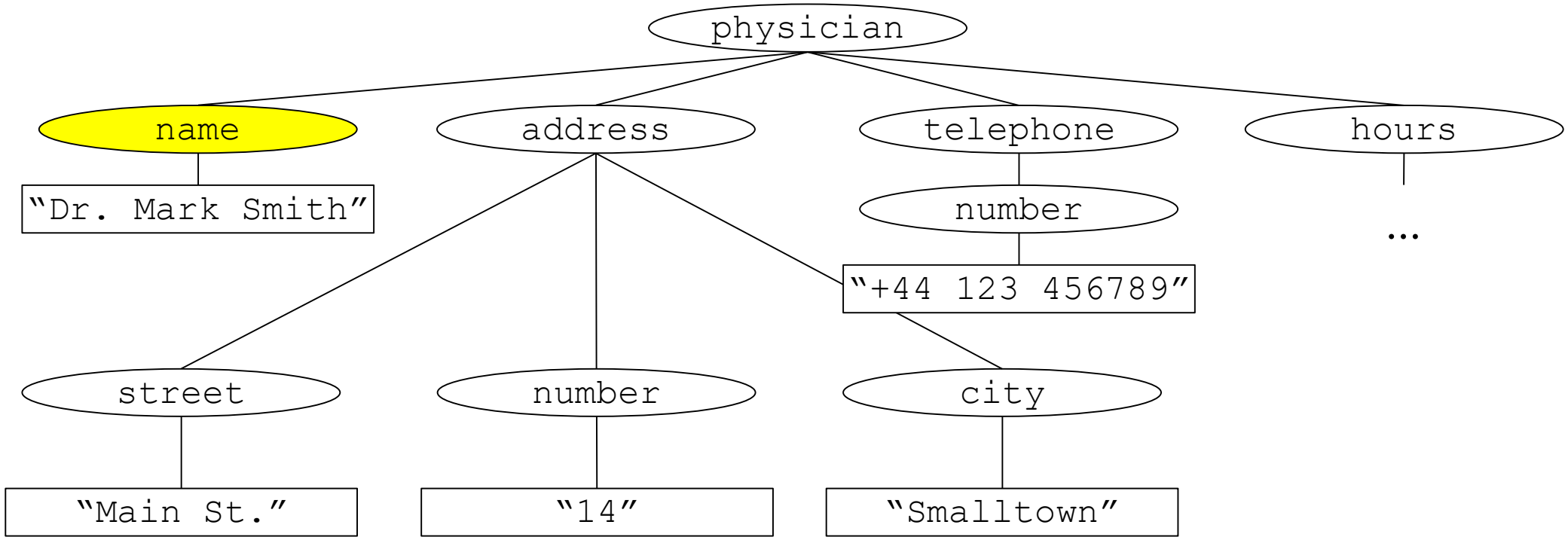
XPath Example 3: Select all Child Element Nodes

- Example: `/physician/*/number`
- Asterisk (*) can be any element



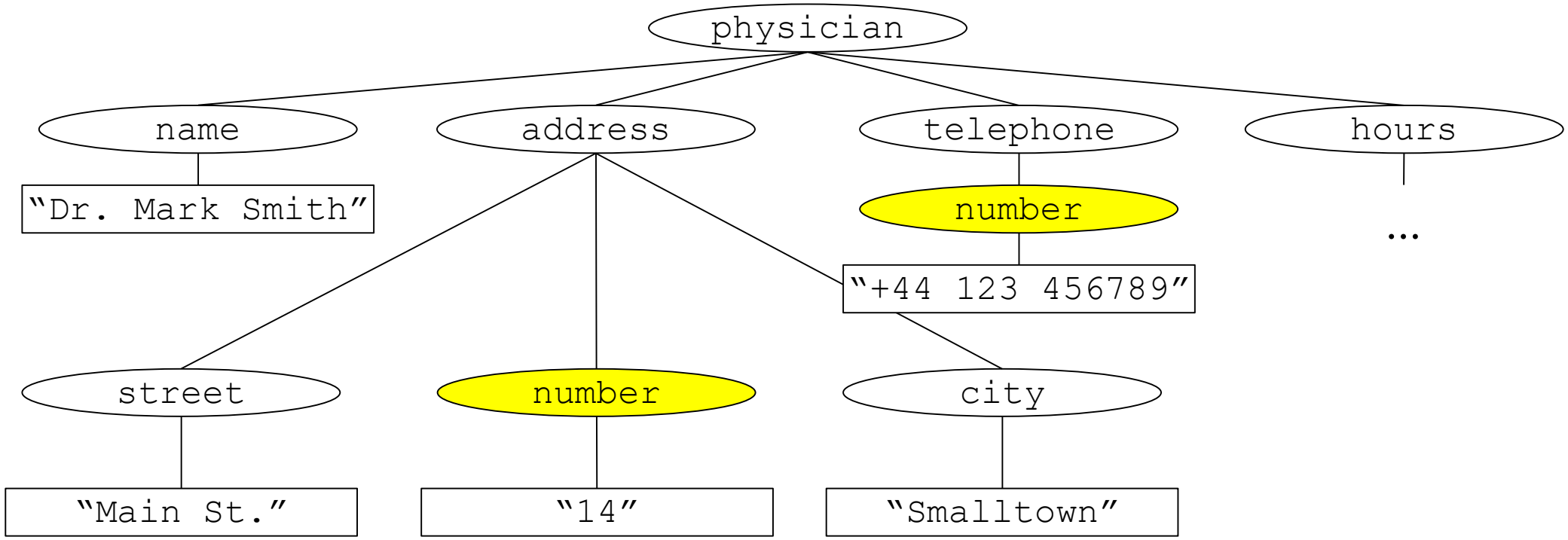
XPath Example 3: Using the Order of Elements

- Example: `/physician/*[1]`
- *[1] returns the first descendant with whatever name
- Note: The tree is ordered!



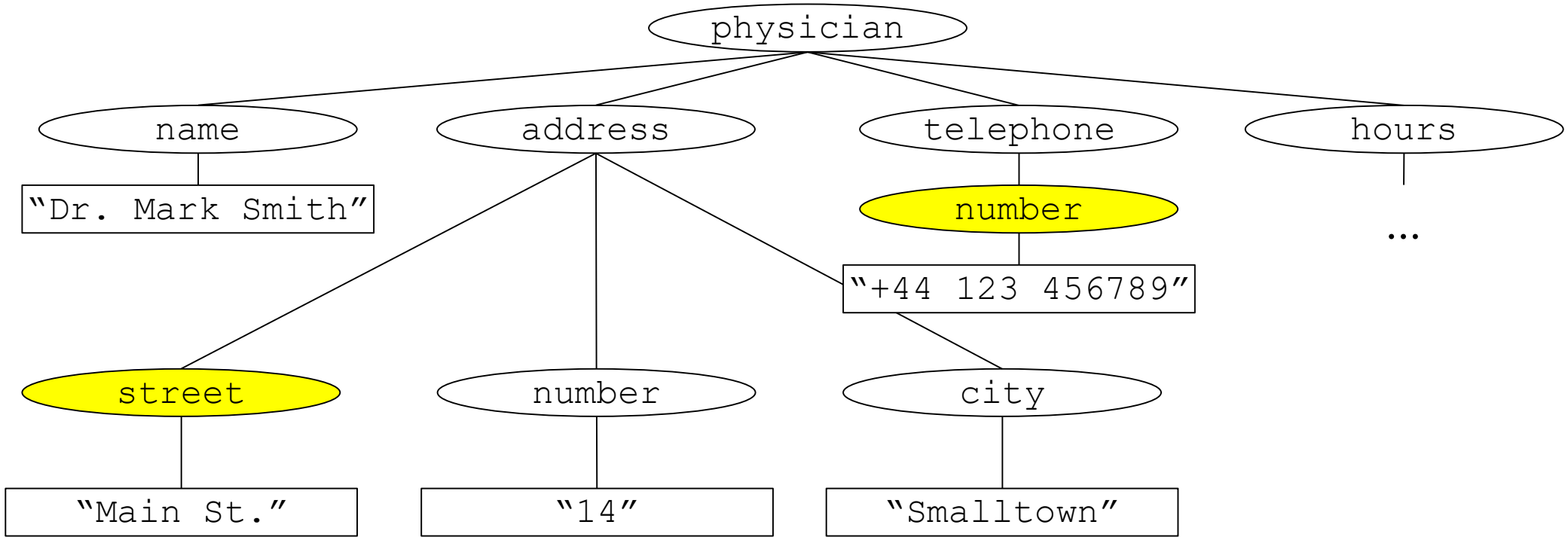
XPath Example 4: Selecting on different Depths

- Example: `/physician//number`
- // stands for an arbitrary line of descendants
- Selected elements can be on different depths in the tree

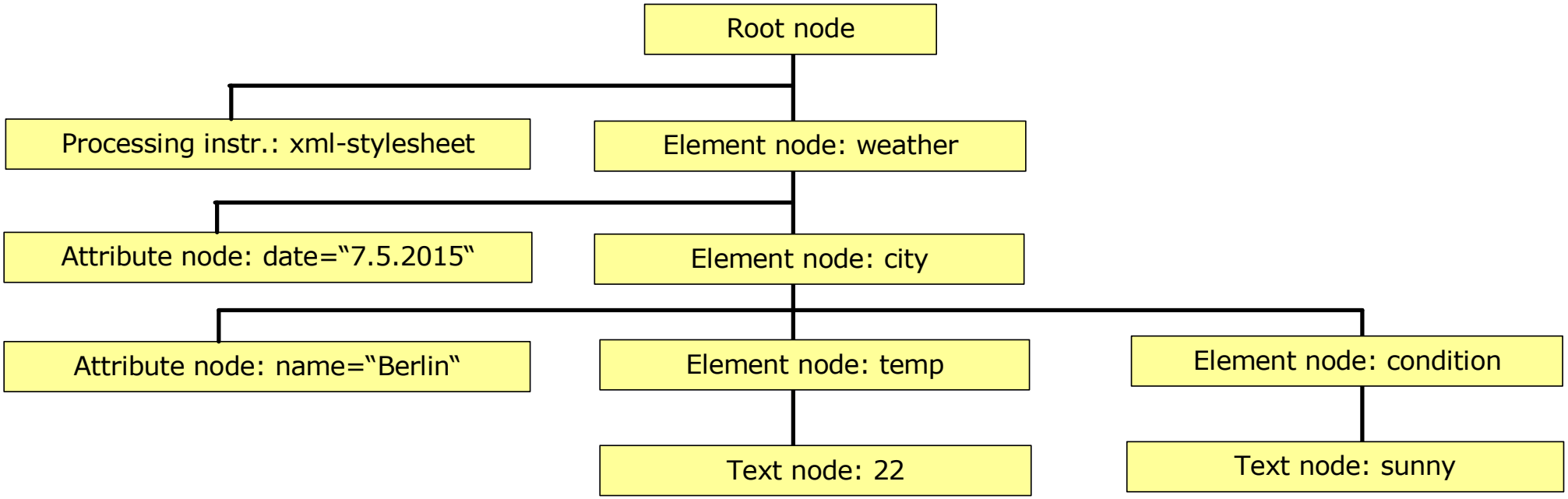


XPath Example 5: Moving Upwards in the Tree

- Example: `/physician//number/../*[1]`
- .. goes up one level



Exercise 1: XPath



Which nodes sets are selected by the following Xpath expressions?

1. /weather/city/temp
2. /weather/city/@name
3. /weather/city/*
4. //condition/text()
5. //condition/../../@date
6. //@date | /weather/city/@name

Predicates

- Predicates allow you to further restrict the selection
- Predicates are expressed using [ ]

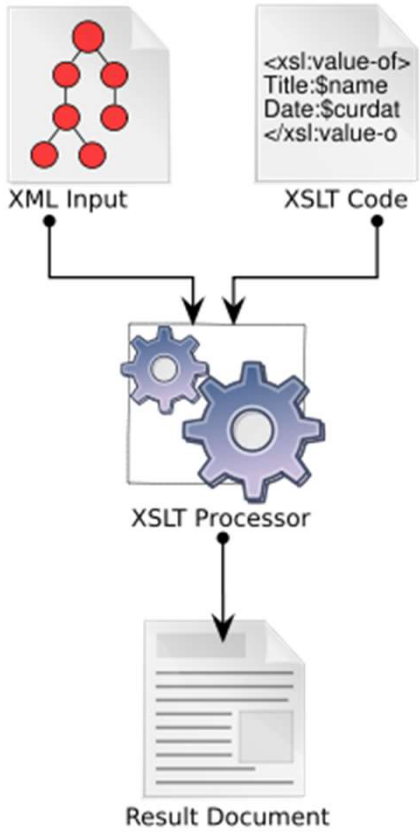
Predicate	Explanation
city[2]	Select second city child element according to order
city[@name = “Berlin"]	Select only city elements which have a name attribute with the value Berlin
city[@name != “Berlin"]	Select only city elements which not have a name attribute with the value Berlin
temp[text()>22]	Select only temp elements with a value exceeding 22
temp *[text()>22 and text()<30]*	Select only temp elements with a values between 22 and 30

- /weather/city[@name = “Berlin"]/temp/text() Result: 22
- /weather/city[1]/temp/text() Result: 22

3.5 eXtensible Stylesheet Language Transformation (XSLT)

Template language for transforming XML documents.

- Used in data integration for translating between different XML formats
- Basic idea:
  1. Pick values out of XML documents using XPath
  2. build new documents from them using templates
     - other XML files
     - HTML files
     - text files, …
- XSLT is a Turing complete language
  - Tutorial: https://www.w3schools.com/xml/xsl_intro.asp
- MapForce can generate XSLT from correspondences



XSLT Templates

- XSLT: Template

```
<xsl:template match="/physician">
  <html>
    <body>
      <h2><xsl:value-of select="name"/></h2>
    </body>
  </html>
</xsl:template>
```

The match attribute is used to associate a template with an XML element.

xsl:value-of is used to extract a value from the XML document and fill it into the template.

- Output:

Nesting XSLT Templates

- Templates can be nested with xsl:apply-templates

```
<xsl:template match="/physician">
  <html>
    <body>
      <h2><xsl:value-of select="name"/></h2>
        <xsl:apply-templates select="address"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="address">
  <h3>Address:</h3>
  <p><xsl:value-of select="street"/>
     <xsl:value-of select="number"/></p>
  <p><xsl:value-of select="city"/></p>
</xsl:template>
```

- Output

Alternative: Looping through all Child Elements

```
<xsl:template match="/">
  <html><body><h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32"><th>Title</th><th>Artist</th></tr>
    <xsl:for-each select="catalog/cd">
    <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body></html>
</xsl:template>
```

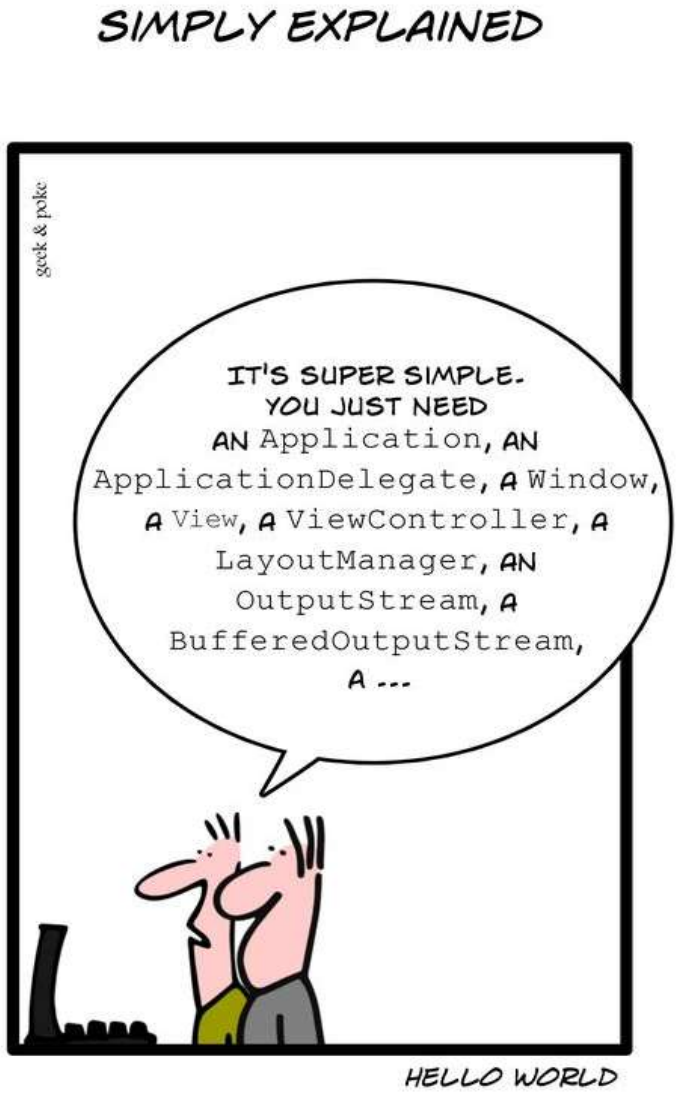
xsl:for-each loop

Sort elements in loop by artist name

Source: http://www.w3schools.com/

3.6 XML in Java

- JAXP: Java API for XML Processing
- Provides for:
  - parsing and validating XML documents
  - DOM interface for accessing nodes using Xpath
  - transforming documents with XSLT
- Included since Java 1.4
- Tutorial: http://docs.oracle.com/javase/tutorial/jaxp/



http://geekandpoke.typepad.com/geekandpoke/2010/06/hello-world.html

Example: Validating against a DTD in Java

- As simple as that:

```
DocumentBuilderFactory factory =
     DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse("data/data.xml");
```

- What happens:

```
Error: URI=file://.../data.xml Line=21:
  The content of element type "physician" must match
  "(name,address*,telephone?,fax,hours)".
```

Example: Using XPath with Java

- Loading an XML document:

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse("data/data.xml");
```

- Defining an XPath Expression:

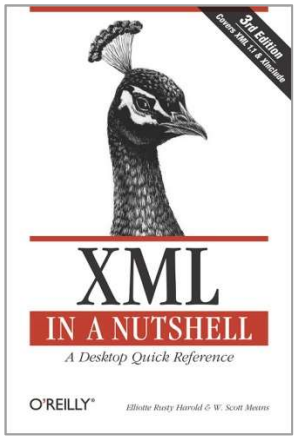
```
XPathFactory xPathFactory = XPathFactory.newInstance();
XPath xpath = xPathFactory.newXPath();
XPathExpression expr = xpath.compile("/physician/name");
```

- Using an XPath Expression:

```
String name = expr.evaluate(doc);
System.out.println(name);
```

References and Experimentation

- Books
  - Harold & Means: XML in a Nutshell. O'Reilly
- Tutorials
  - Character encoding: http://www.cs.tut.fi/~jkorpela/chars.html
  - XML: https://www.w3schools.com/xml/
  - XPath: https://www.w3schools.com/xml/xpath_intro.asp
  - JAXP: http://docs.oracle.com/javase/tutorial/jaxp/
- Tools
  - Altova XMLSpy: Powerful XML Editor supporting a wide range of XML technologies: http://www.altova.com/xmlspy.html
  - XML Notepad: Simple XML Editor for Windows: https://github.com/microsoft/xmlnotepad



Outline

1. Data Exchange Formats - Part I
    1. Character Encoding
    2. Comma Separated Values (CSV)
        1. Variations
        2. CSV in Java
    3. Extensible Markup Language (XML)
        1. Basic Syntax
        2. DTDs
        3. Namespaces
        4. XPath
        5. XSLT
        6. XML in Java
2. Data Exchange Formats - Part II
    1. JavaScript Object Notation (JSON)
    2. Resource Description Framework (RDF)