XML intro

What is XML?

- XML stands for E**X**tensible **M**arkup **L**anguage
- XML is a **markup language** much like HTML
- XML was designed to **carry data**, not to display data
- XML tags are not predefined. You must **define your own tags**
- XML is designed to be **self-descriptive**
- XML is a **W3C Recommendation**

The Difference Between XML and HTML

- XML is not a replacement for HTML.

- XML and HTML were designed with different goals:
  - XML was designed to transport and store data, with focus on what data is. (like model)
  - HTML was designed to display data, with focus on how data looks. (like view)

  Therefore - HTML is about displaying information, while XML is about carrying information.

XML Does not DO Anything

- XML was created to structure, store, and transport information.
- The following **example**
  is a note to Tove from Jani, stored as XML:

-
  ```
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this Weekend!</body>
  </note>
  ```
- The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.
- But still, this XML document does not DO anything.

XML Simplifies Data Sharing

- In the real world, computer systems and databases contain data in incompatible formats.
- XML data is stored in plain text format. This provides a software- and hardware-independent way of storing and exchanging data.
- This makes it much easier to create data that different applications can share.

XML Simplifies Data Transport

- With XML, data can easily be exchanged between incompatible systems.
- One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
- Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Documents Form a Tree Structure

- XML documents must contain a **root element**. This element is "the parent" of all other elements.

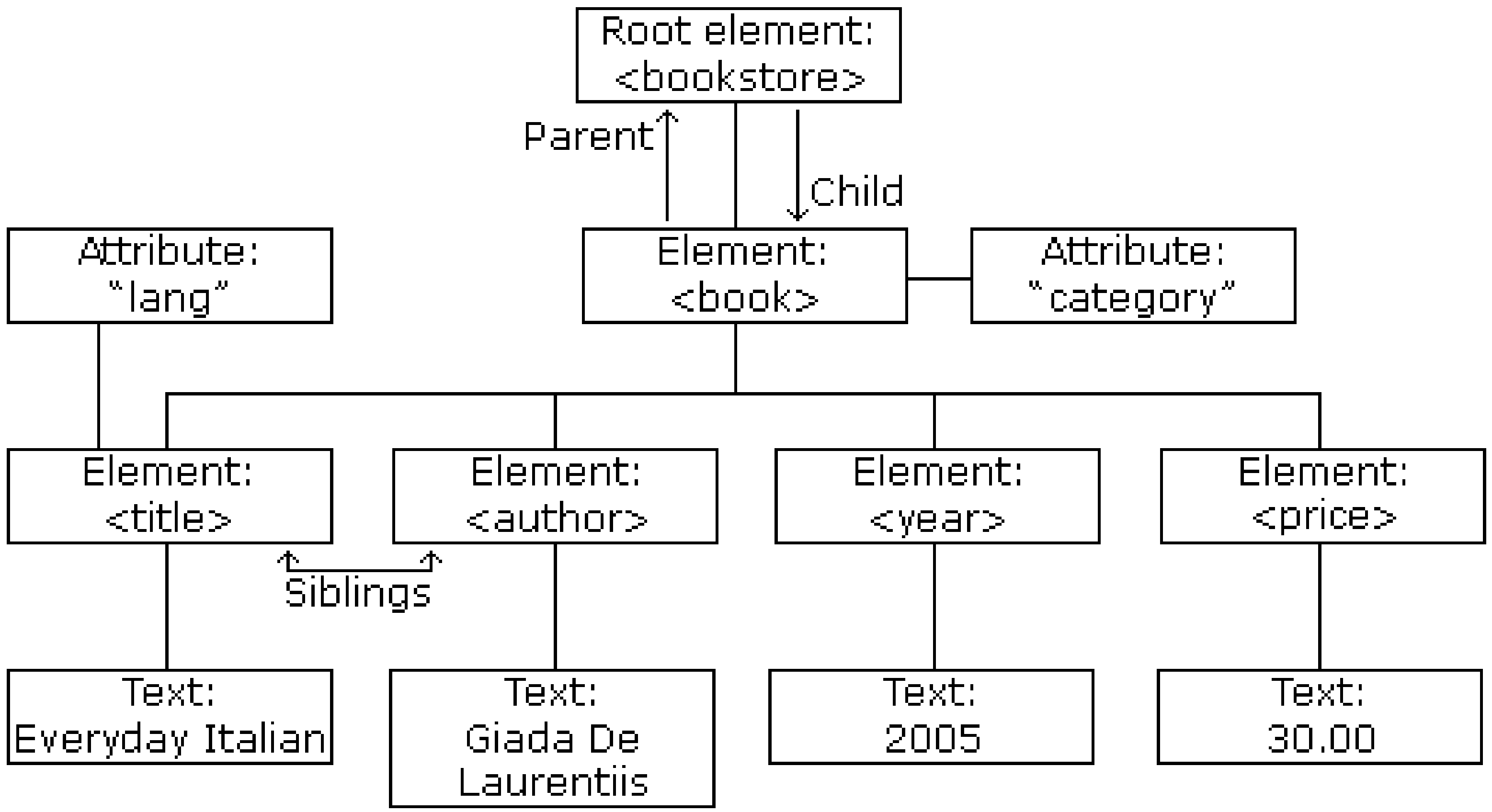
- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

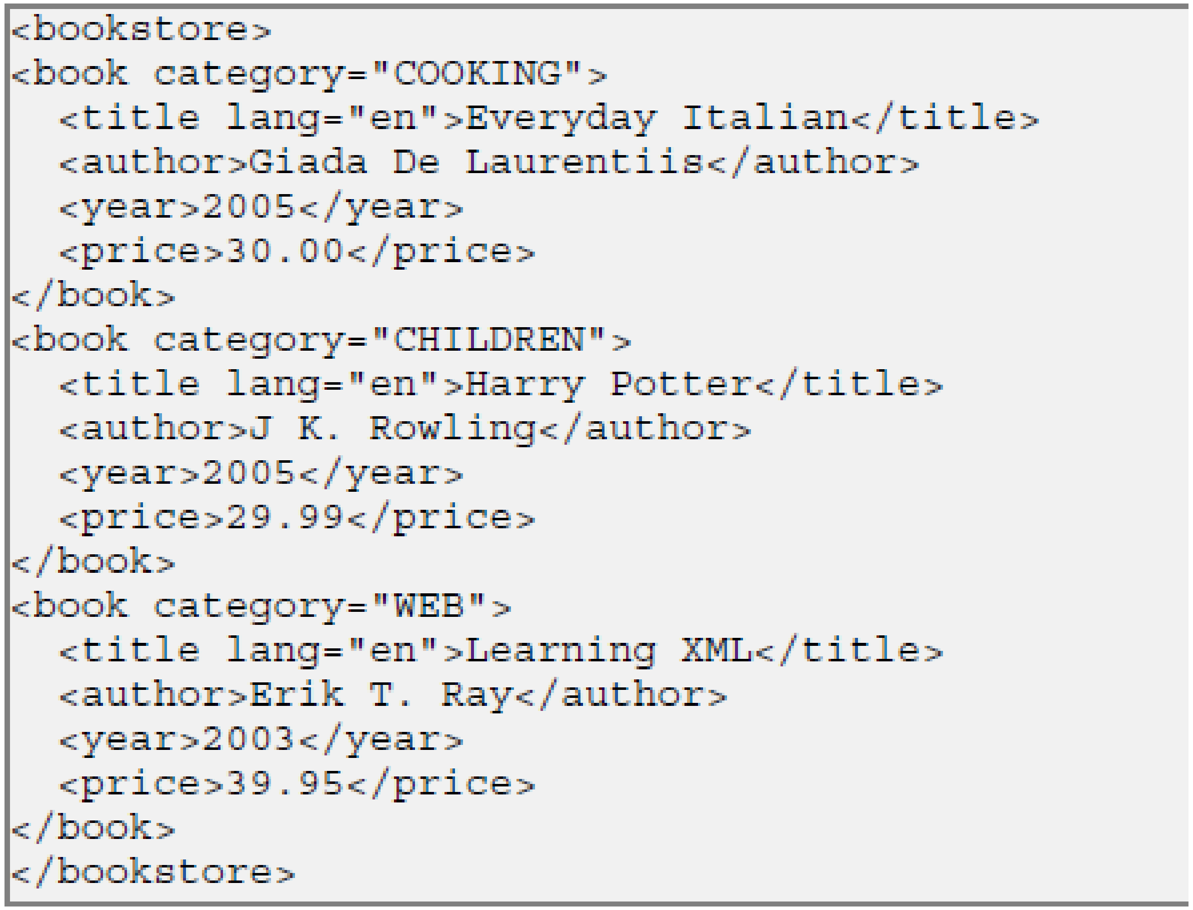
XML Documents - example

- All elements can have sub elements (child elements):

-
  ```
  <root>
    <child>
      <subchild>.....</subchild>
    </child>
  </root>
  ```
- The terms parent, child, and sibling are used to describe the relationships between elements.
- Parent elements have children. Children on the same level are called siblings (brothers or sisters).

Example of XML-dom-tree





```
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.

The <book> element itself has 4 children: <title>,< author>, <year>, <price>.

XML Syntax Rules – to be wellformed

- **All XML Elements Must Have a Closing Tag**
- **XML Tags are Case Sensitive**
- **XML Documents must have <u>one</u> Root Element**
- **XML Elements must be Properly Nested**
- **XML Attribute values must be Quoted**
- **Entity References**

XML Syntax Rules 2

- **Entity References**
  - Some characters have a special meaning in XML.
  - If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
  - Example - This will generate an XML error:
    `<message>if salary < 1000 then</message>`
    To avoid this error, replace the "<" character with an **entity reference**:

  There are 5 predefined entity references in XML:

– &lt;	<	less than
– &gt;	>	greater than
– &amp;	&	ampersand
– &apos;	’	apostrophe
– &quot;	”	quotation mark

- **Note:** Only the characters "<" and "&" are strictly illegal in XML.
  The greater than character is legal, but it is a good habit to replace it.

XML Elements vs. Attributes

- Take a look at these two examples:

```
<person sex="female">      // Attribute
<firstname>Anna</firstname> // Sex inf. to 'person'-tag
<lastname>Smith</lastname>
</person>
```

-
  ```
  <person>                    // Element
  <sex>female</sex>             // Sex separate tag
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
  </person>
  ```

- Both examples provide the same information.

- There are no rules about when to use attributes and when to use elements. But in general use elements **except** for metadata.

Valid XML Documents

- A "Valid" XML document is
  - "Well Formed" XML document
  - Conforms to a Document Type Definition (DTD):
-
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to><from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```
- The DOCTYPE declaration in the example above, is a reference to an external DTD file.

XML DTD

- The purpose of a DTD is to define the structure of an XML document.
  It defines the structure with a list of legal elements:

-
  ```
  <!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
  ]>
  ```
- xxx+ ->1-many   xxx* -> 0-many   xxx? -> 0-1
- , -> and | -> or

XML Schema

- W3C supports an XML based alternative to DTD called XML Schema:
-
```
<xs:element name="note">
<xs:complexType>
 <xs:sequence>
  <xs:element name="to"    type="xs:string"/>
  <xs:element name="from"    type="xs:string"/>
  <xs:element name="heading" type="xs:string"/>
  <xs:element name="body"    type="xs:string"/>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

Json - JavaScript Object Notation

- Language for storing and exchanging data (Like XML)
- Platform independent (like XML)
- Program Language independent (Like XML)
- No validation (unlike XML)
- More compressed notation than XML
  (e.g. Car(model,color,registrationNumber)
  XML = 235 Char, Json=56)

Json - structure

- { object }
- “name” : “value”
- “name” : [“value1”, “value2”]

Example

{“Book”: {“title”:”Applying UML”, “Author”:”Larman”}}