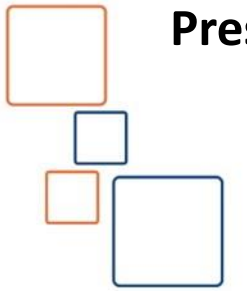
XML and JSON Essentials with Java



Presented By:

Mona Mahrous, MSc

Invest In Yourself,
Develop Your Career

Course Outlines

- **Chapter (1):** XML

  What is XML? What XML can do? How to write XML?

- **Chapter (2):** JSON

  What is JSON? What JSON can do? How to write JSON?

- **Chapter (3):** JSON APIs

  Used to parse JSON using Java API

Java™ Education and Technology Services

Chapter 1

XML

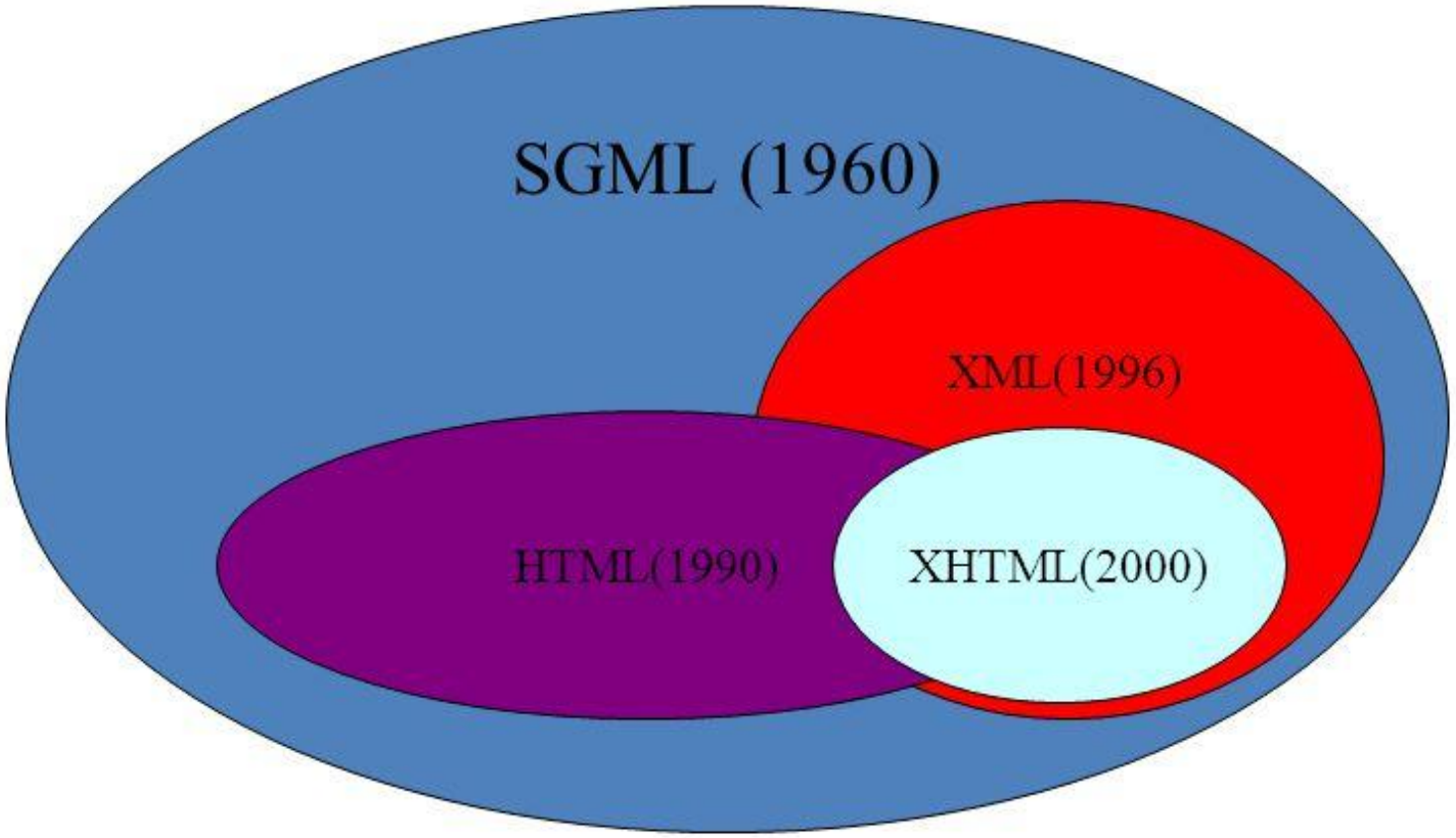
What, Why & How?

Ch1 : What is XML?

Java™ Education and Technology Services

- Stands for e**X**tensible **M**arkup **L**anguage.
- XML is a markup language much like HTML used to describe data.
- W3C Recommendation, since February 1998.
- XML was designed to be both human- and machine-readable.

History: SGML vs. HTML vs. XML



http://www.w3.org/TR/2006/REC-xml-20060816/

Ch1 : XML Features

- XML is easy to understand.
- It is platform independent.
- XML was designed to ***store*** and ***transport*** data.
  - **Store Data**
    - XML Databases
      - XML-Enabled (Oracle, MS SQL Server, DB2, …)
      - Native-XML (BaseX, eXist, …)
    - User Interface Design (Qt, JavaFX, Android, …)
    - Configuration Files (Frameworks, Libraries, Mapping, …)
    - Settings Files(IDEs, Programs, Games, …)
  - **Transport Data**
    - Web Services (SOAP, REST, …)

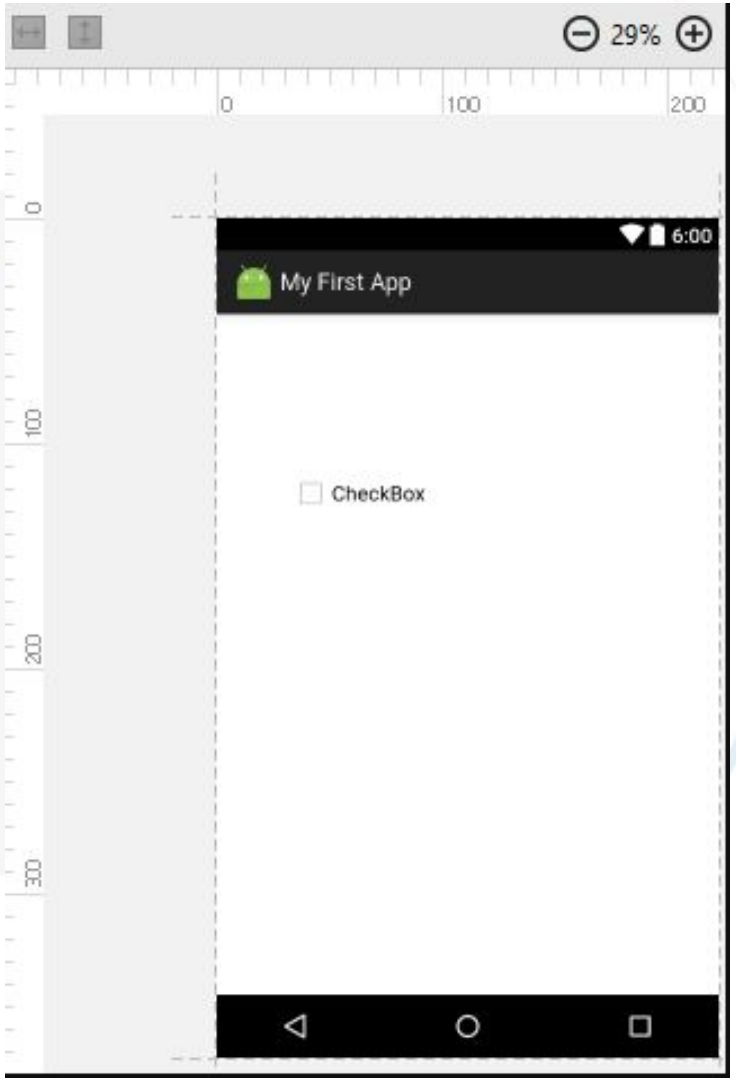
Ch1 : Why XML?

- **Android Views:**

`view_customs.xml` to be:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/custom_view"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >

    <TextView android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        />
</LinearLayout>
```



Ch1 : Why XML?

Java™ Education and Technology Services

- **iOS Storyboard:**

```
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="14490.70" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES">
    <device id="retina6_1" orientation="portrait">
        <adaptation id="fullscreen"/>
    </device>
    <dependencies>
        <deployment identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="14490.49"/>
        <capability name="Safe area layout guides" minToolsVersion="9.0"/>
        <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
    </dependencies>
    <scenes>
        <!--View Controller-->
        <scene sceneID="SAQ-91-eGP">
            <objects>
                <viewController id="sRn-T1-EGi" sceneMemberID="viewController">
                    <view key="view" contentMode="scaleToFill" id="xQm-XW-XAG">
                        <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <subviews>
                            <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="etx-p4-ZLn">
                                <rect key="frame" x="80" y="674" width="254" height="44"/>
                                <color key="backgroundColor" red="0.0" green="0.5" blue="0.91494277400000001" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                                <constraints>
                                    <constraint firstAttribute="height" constant="44" id="8sH-Ou-My2"/>
                                </constraints>
                                <state key="normal" title="Login">
                                    <color key="titleColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>
                                </state>
                                <connections>
                                    <segue destination="jG5-aa-SSP" kind="presentation" id="fJK-lO-pcU"/>
                                </connections>
                            </button>
                        </subviews>
```

Ch1 : Why XML?

Java™ Education and Technology Services

- **Maven and POM.xml:**

android-maven-example / pom.xml

Code | Blame | 132 lines (118 loc) · 4.86 KB

```
12
13          <modules>
14              <module>androidexample-lib</module>
15              <module>androidexample-app</module>
16              <module>androidexample-tests</module>
17          </modules>
18
19
20          <dependencyManagement>
21              <dependencies>
22                  <dependency>
23                      <groupId>com.google.android</groupId>
24                      <artifactId>android</artifactId>
25                      <version>4.1.1.4</version>
26                      <scope>provided</scope>
27                  </dependency>
28
29                  <dependency>
30                      <groupId>com.google.android</groupId>
31                      <artifactId>android-test</artifactId>
32                      <version>4.1.1.4</version>
33                      <scope>provided</scope>
34                  </dependency>
35
36              </dependencies>
37          </dependencyManagement>
38
39          <build>
```

Ch1 : XML Example

Java™ Education and Technology Services

Aya, Ahmed , Reminder, Don’t forget me …

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

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

Ch1 : XML Does Not Do Anything

Java™ Education and Technology Services

- Maybe it is a little hard to understand, but XML does not DO anything.

- The XML above is quite self-descriptive:
  - It has sender information.
  - It has receiver information
  - It has a heading
  - It has a message body.
  - But still, the XML above does not DO anything.
  - XML is just information wrapped in tags.

- Someone must write a piece of software to ***send***, ***receive***, ***store***, or ***display*** it.

Ch1 : Self-Describing Syntax

Java™ Education and Technology Services

- A prolog defines the XML version and the character encoding

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

- The next line is the root element of the document

```
<bookstore>
```

- The next line starts a <book> element
  - The <book> elements have 4 child elements:
    <title>, <author>, <year>, <price>

```
<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
```

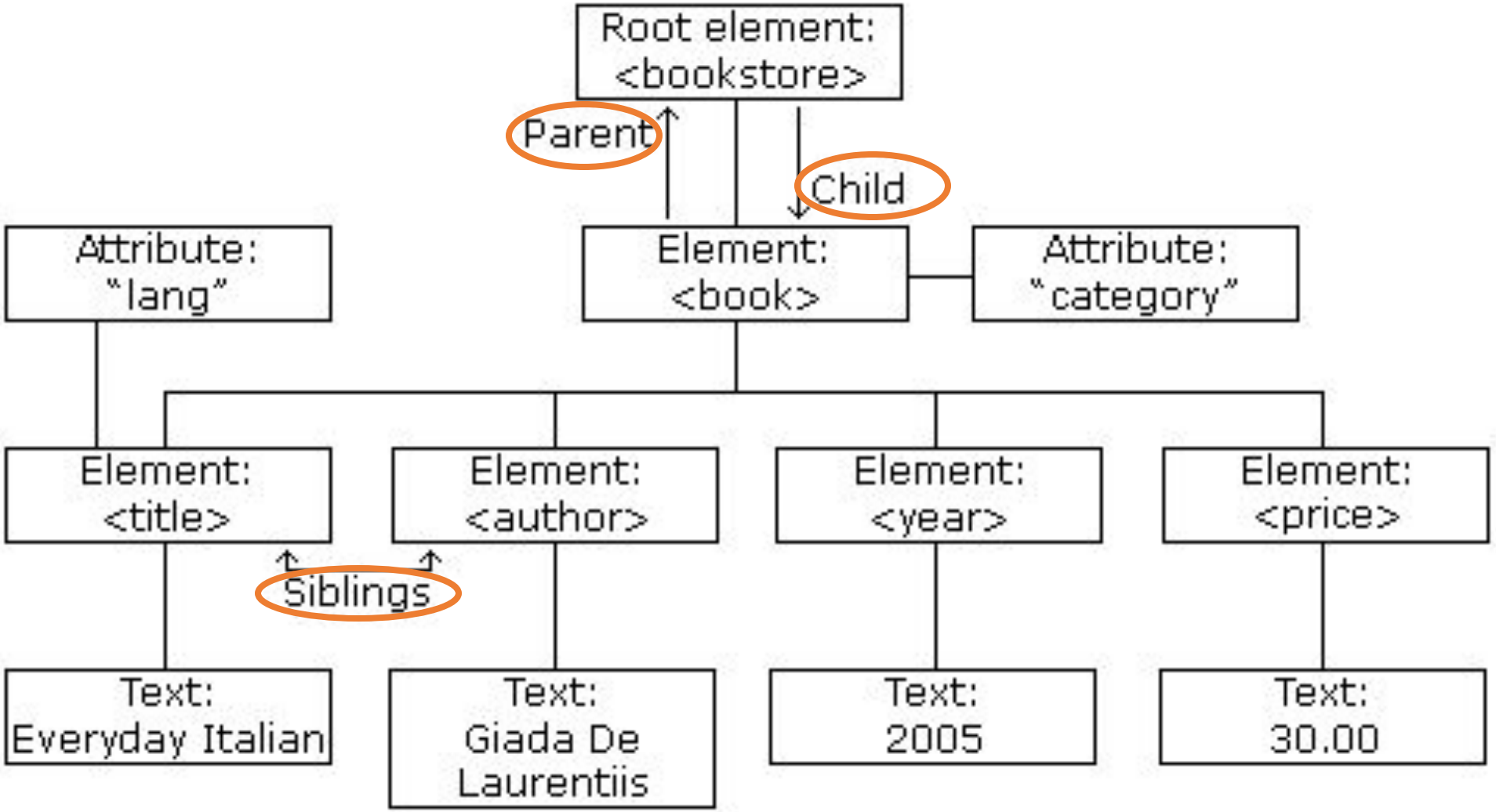
- The next line ends the book element

```
</book>
```

- The next line ends the root element of the document

```
</bookstore>
```

Ch1 : XML Tree



Ch1 : XML Tree

Java™ Education and Technology Services

- XML documents form a tree structure that starts at "the root" and branches to "the leaves".
- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- 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.
- Parents have children. Children have parents.
- Siblings are children on the same level (brothers and sisters).
- All elements can have text content and attributes.

Ch1 : XML Syntax Rules

Java™ Education and Technology Services

- XML Documents Must Have a Root Element
- The XML Prolog
  `<?xml version="1.0" encoding="UTF-8"?>`
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted

Ch1 : XML Syntax Rules

Java™ Education and Technology Services

- Entity References
- CDATA Sections
- Comments in XML
- White-space is Preserved in XML
- Well Formed XML

Ch1 : XML Documents Must Have a Root Element

- XML documents must contain one root element that is the parent of all other elements:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- In this example <note> is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Aya</to>
  <from>Amin</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Ch1 : The XML Declaration (Prolog)

Java™ Education and Technology Services

- This line is called the XML prolog:

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

- The XML prolog is optional. If it exists, it must come first in the document.
- XML documents can contain international characters, like Norwegian øæå or French êèé.
- To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.
- UTF-8 is the default character encoding for XML documents.

- UTF-8 is also the default encoding for HTML5, CSS, JavaScript, PHP, and SQL.

Ch1 : Syntax Rules for XML Declaration (Prolog)

Java™ Education and Technology Services

- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important.
  The correct order is: version, encoding and standalone.
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

Ch1 : XML declaration

Java™ Education and Technology Services

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
```

Parameter	Parameter_value	Parameter_description
Version	**1.0** or **1.1**	Specifies the version of the XML standard used. ***(The Only Mandatory Part)***
Encoding	**UTF-8**, UTF-16, ISO-8859-1, Windows-1251, …	It defines the character encoding used in the document. ***UTF-8 is the default encoding used.***
Standalone	**yes** or **no**	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. ***The default value is set to no*****.** Setting it to yes tells the processor there are no external declarations required for parsing the document.

```
<?xml version=
standalone="n
<!DOCTYPE no
<note>
  <to>Alice</to
</note>
```

Ch1 : All XML Elements Must Have a Closing Tag

- In XML, it is illegal to omit the closing tag. All elements must have a closing tag:

```
<paragraph>This is a paragraph.</paragraph>

<line-break />
```

self closing tag for nobody elements

- Note: The XML prolog does not have a closing tag! This is not an error.
- The prolog is not a part of the XML document.

The prolog is **metadata** (instructions for the parser), not data.

Ch1 : XML Tags are Case Sensitive

Java™ Education and Technology Services

- XML tags are case sensitive.
- The tag <Letter> is different from the tag <letter>.
- Opening and closing tags must be written with the same case:

```
<message>This is correct</message>

<Message>This is NOT correct</message>
```

- "**Opening** and **Closing** tags" are often referred to as "**Start** and **End** tags".
- Use whatever you prefer.
- It is exactly the same thing.

Ch1 : XML Elements Must be Properly Nested

- In HTML, you might see ***improperly nested*** elements:

```
<b> <i> This text is bold and italic </b> </i>
```

- In XML, all elements ***must be properly nested*** within each other:

```
<b> <i> This text is bold and italic </i> </b>
```

- In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

Ch1 : XML Attribute Values Must Always be Quoted

- XML elements can have attributes in name/value pairs just like in HTML.
- In XML, the attribute values must always be quoted:

```
<note date="12/11/2020">
  <to>Aya</to>
  <from>Amin</from>
</note>
```

```
<note date='12/11/2020'>
  <to>Aya</to>
  <from>Amin</from>
</note>
```

Ch1 : Entity References

Java™ Education and Technology Services

- 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.

- This will generate an XML *error*:

```
<message>salary <  1000</message>
```

- To avoid this error, replace the "<" character with an entity reference:

```
<message>salary &lt; 1000</message>
```

- Entity References: Begin with ampersand (&) and end with semicolon (;)

Ch1 : Entity References

Java™ Education and Technology Services

- There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

- Only < and & are strictly illegal in XML, but it is a good habit to replace > with > as well.

Ch1 : CDATA Sections

Java™ Education and Technology Services

- May contain text, reserved characters and white space
  - Reserved characters need not be replaced by entity references

- Not processed by XML parser

- Commonly used for scripting code (e.g., JavaScript)

- Begin with : `<![CDATA[`

- Terminate with : `]]>`

Ch1 : CDATA Example

```
1 <?xml version = "1.0"?>
2
3 <!-- Fig. 5.7 : cdata.xml                    -->
4 <!-- CDATA section containing C++ code      -->
5
6 <book title = "C++ How to Program" edition = "3">
7
8    <sample>
9          // C++ comment
10         if ( this-&gt;getX() &lt; 5 &amp;&amp; value[ 0 ] != 3
11            cerr &lt;&lt; this-&gt;displayError();
12   </sample>
13
14   <sample>
15      <![CDATA[
16
17         // C++ comment
18         if ( this->getX() < 5 && value[ 0 ] != 3 )
19            cerr << this->displayError();
20      ]]>
21   </sample>
22
23   C++ How to Program by Deitel &amp; Deitel
24</book>
```

Entity references required if not in **CDATA** section

XML does not process **CDATA** section

Note the simplicity offered by **CDATA** section

Aya, Ahmed , 2

Ch1 : Comments in XML

Java™ Education and Technology Services

- The syntax for writing comments in XML is similar to that of HTML:

```
<!-- This is a comment -->
```

- Two dashes in the middle of a comment are not allowed:

```
<!-- This is an invalid -- comment -->
```

Ch1 : White-space is Preserved in XML

Java™ Education and Technology Services

- XML does not truncate multiple white-spaces
  (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello Aya
HTML:	Hello Aya

Ch1 : XML Stores New Line as LF

- Windows applications store a new line as: carriage return and line feed (CR+LF).
- Unix and Mac OSX use LF.
- Old Mac systems use CR.

- **XML** stores a new line as **LF**.

Ch1 : Well-formed XML

XML document Considered ***well formed*** if it has:

1. Single root element.
2. Each element has start tag and end tag.
   - Empty element is defined as: <element/>
3. Tags well nested.
   - Incorrect: `<x><y>hello</x></y>`
   - Correct: `<x><y>hello</y></x>`
4. Attribute values in quotes(single or double).
5. Tag & Attributes names written as variable names:
   - Start with character,
   - One word “must not contain spaces”,
   - Case sensitive.
6. An element may not have two attributes with the same name.

Ch1 : XML Elements

Java™ Education and Technology Services

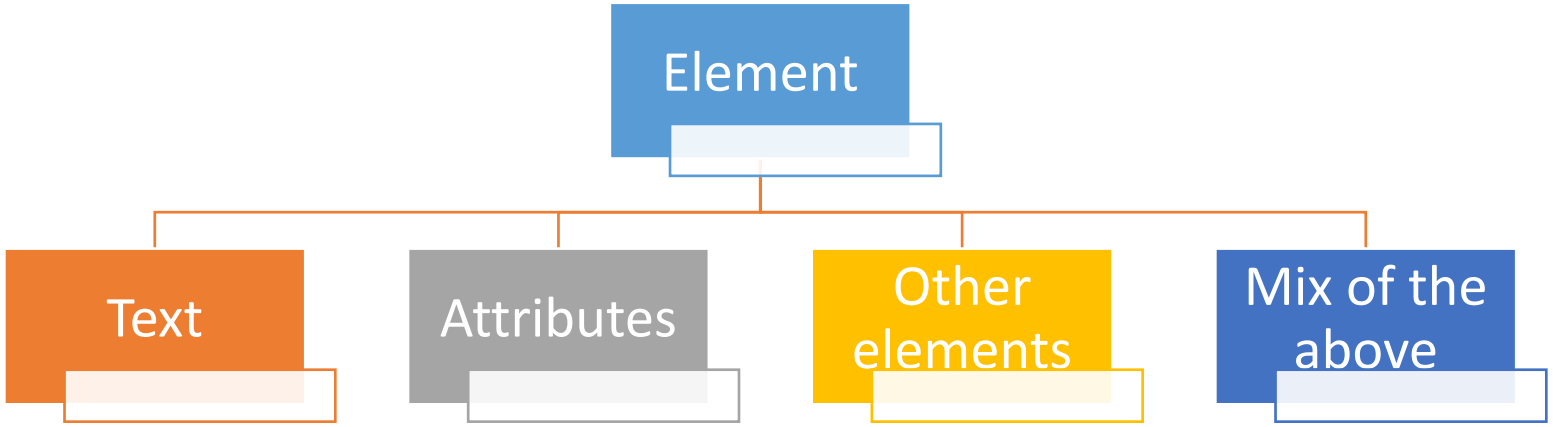
- What is an XML Element?
- Empty XML Elements
- XML Naming Rules
- Best Naming Practices
- Naming Styles

Ch1 : What is an XML Element?

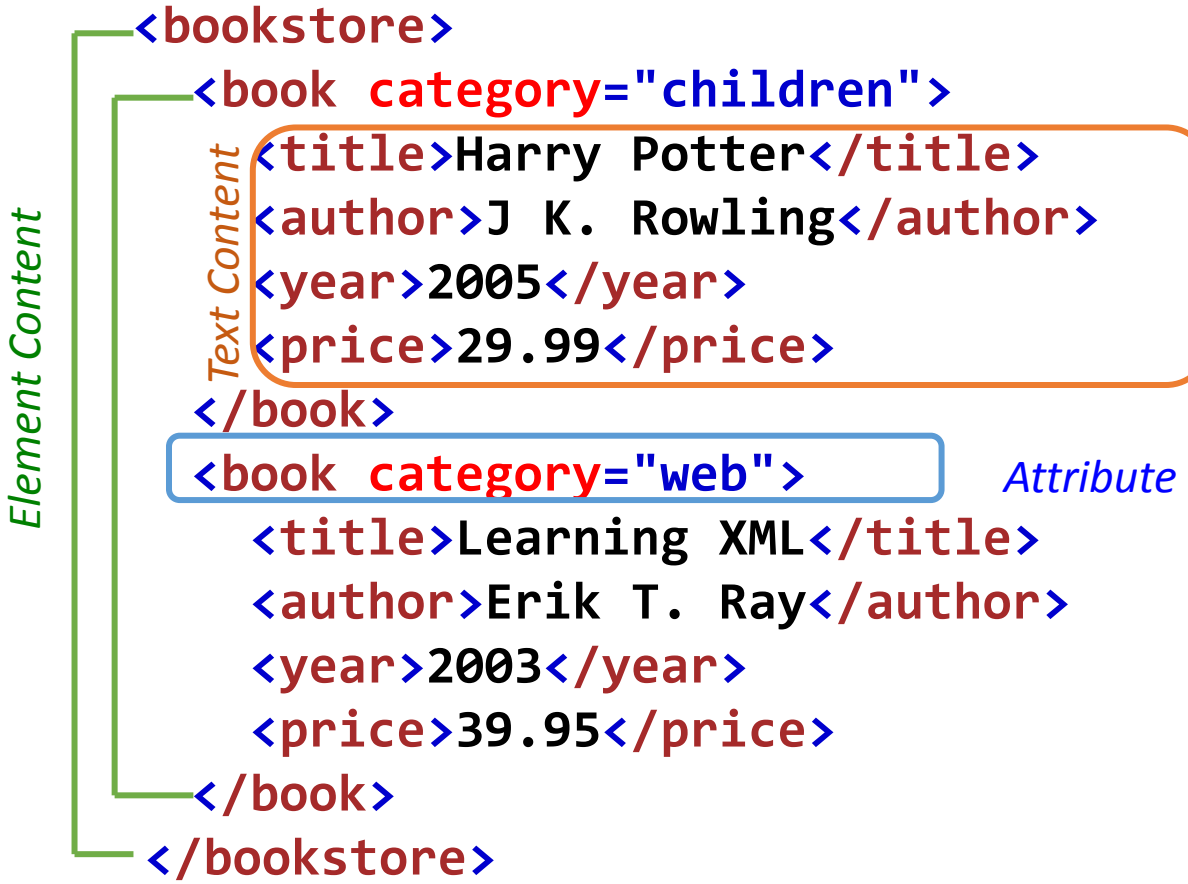
Java™ Education and Technology Services

- An XML document contains XML Elements.
- An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```



Ch1 : What is an XML Element?



Ch1 : What is an XML Element?

Java™ Education and Technology Services

- In this example:

- <title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99).

- <bookstore> and <book> have **element contents**, because they contain elements.

- <book> has an **attribute** (category="children").

Ch1 : Empty XML Elements

Java™ Education and Technology Services

- An element with no content is said to be empty.
- Empty elements can have attributes.

```
<element></element>
```

```
<element />
(Self-Closing)
```

- The two forms produce identical results in XML software (Readers, Parsers, Browsers).

Ch1 : XML Naming Rules

Java™ Education and Technology Services

- XML elements must follow these naming rules:
  - **Element names**

<first name>

- are case-sensitive
- must start with a letter or underscore
- can contain letters, digits, hyphens, underscores, and periods
- cannot start with the letters xml (or XML, or Xml, etc)
- cannot contain spaces

Ch1 : Best Naming Practices

Java™ Education and Technology Services

- Create descriptive names, like this: <person>, <firstname>, <lastname>.

- Create short and simple names,
  like this: <book_title> not like this: <the_title_of_the_book>.

- Avoid "-" If you name something "first-name",
  some software may think you want to subtract "name" from "first".

- Avoid "." If you name something "first.name",
  some software may think that "name" is a property of the object "first".

- Avoid ":" Colons are reserved for namespaces (more later).

- Non-English letters like éòá are perfectly legal in XML,
  but watch out for problems if your software doesn't support them.

Ch1 : Naming Styles

Java™ Education and Technology Services

- There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

- If you choose a naming style, it is good to be consistent!
- XML documents often have a corresponding database.
- A common practice is to use the naming rules of the database for the XML elements.
- Camel case is a common naming rule in Java, JavaScript.

Ch1 : XML Attributes

- XML Attributes Must be Quoted
- XML Elements vs. Attributes
- My Favorite Way
- Avoid XML Attributes?
- XML Attributes for Metadata

Ch1 : XML Attributes Must be Quoted

Java™ Education and Technology Services

- XML elements can have attributes, just like HTML.
- Attributes are designed to contain data related to a specific element.
- Attribute values must always be quoted. Either single or double quotes can be used.

Ch1 : XML Attributes Must be Quoted

Java™ Education and Technology Services

- For a person's gender, the <person> element can be written like this:

```
<person gender="female">
```

- or like this:

```
<person gender='female'>
```

- If the attribute value itself contains double quotes you can use single quotes, like this:

```
<gangster name='George "Shotgun" Ziegler'>
```

- or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

Ch1 : XML Elements vs. Attributes

Java™ Education and Technology Services

- Take a look at these examples:

```
<person gender="female">
  <firstname>Aya</firstname>
  <lastname>Amin</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Aya</firstname>
  <lastname>Amin</lastname>
</person>
```

- In the first example gender is an attribute.
- In the last, gender is an element. Both examples provide the same information.

- There are no rules about when to use attributes or when to use elements in XML.

Ch1 : How we do it:

Java™ Education and Technology Services

- The following three XML documents contain exactly the same information:

- A date attribute is used in the first example:
- A <date> element is used in the second example:
- **An expanded <date> element is used in the third example (*Preferred*):**

```
<note date="2020-12-30">
  <to>Aya</to>
  <from>Amin</from>
</note>
```

```
<note>
  <date>2020-12-30</date>
  <to>Aya</to>
  <from>Amin</from>
</note>
```

```
<note>
  <date>
    <year>2020</year>
    <month>12</month>
    <day>30</day>
  </date>
  <to>Aya</to>
  <from>Amin</from>
</note>
```

Ch1 : Avoid XML Attributes?

Java™ Education and Technology Services

- Some things to consider when using attributes are:

  ✔ attributes cannot contain multiple values (elements can)
  ✔ attributes cannot contain tree structures (elements can)
  ✔ attributes are not easily expandable (for future changes)

- **Don't** end up like this:

```
<note day="30" month="12" year="2020"
to="Aya" from="Amin" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

Ch1 : XML Attributes for Metadata

Java™ Education and Technology Services

- Sometimes ID references are assigned to elements.
- These IDs can be used to identify XML elements in much the same way as the id attribute in HTML.
- This example demonstrates this:

```
<messages>

  <note id="501">
    <to>Aya</to>
    <from>Amin</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>

  <note id="502">
    <to>Aya</to>
    <from>Amin</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>

</messages>
```

- The id attributes above are for identifying the different notes. It is not a part of the note itself.
- **What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.**

Ch1 : XML Parser

Java™ Education and Technology Services

- XML parser is a software, library or a package that provides interface for client applications to work with XML documents.
- It checks for proper format of the XML document and may also validate the XML documents.

- XML Parser:
  - Processes XML document
  - Reads XML document
  - Checks syntax
  - Reports errors (if any)

- Example:
  - Internet browsers (Chrome, Firefox, Edge, Internet Explorer, …)
  - XML Editors (XmlSpy, Microsoft XML Notepad, XMLQuire, OxygenXML, …)
  - Built-in components in the Java JDK and several 3rd Party Libraries Jackson

Ch1 : XML Processing Instructions

Java™ Education and Technology Services

- Processing instructions (PIs) allow documents to contain instructions for applications.
- PIs are not part of the character data of the document but MUST be passed through to the application.

- Processing instructions (PIs) can be used to pass information to applications.
- PIs can appear anywhere in the document outside the markup.
- They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

- A PI starts with a special tag **<?** and ends with **?>**.
- Processing of the contents ends immediately after the string **?>** is encountered.

- Syntax: **<?**target instructions**?>**

Lab Exercise

Java™ Education and Technology Services

- **1st Assignment: A Configuration File**
  - Design a configuration file for a library.
    - Info. of library consists of a location, a description of the library, a librarian and a lot of books.
    - Each book has title, ISBN, and Author.
    - The book contains also a preface and many of parts.
    - Each part has title and contains many of chapters.
    - Each chapter has title and contains a summary and many of sections.
    - Sections contain the content of the book as paragraphs.

XML must have elements (usual and empty), attributes, and CDATA section