## Java$^{TM}$ Education & Technology Services

# Introduction to XML

## Presentation Outline

- **<u>Chapter (1):</u>** Introduction To XML

  ***What is XML? and, what XML can do?***

- **<u>Chapter (2):</u>** Creating XML

  ***To learn XML syntax rules (writing XML document)***

- **<u>Chapter (3):</u>** DTD

  ***To validate the XML document.***

- **<u>Chapter (4):</u>** XML Schema

  ***Another way to validate the of XML document.***

# Presentation Outline

- **Chapter (5):** XPath

  ***Used to navigate in an XML document.***

- **Chapter (6):** XSLT

  ***Used to convert XML into another format.***

- **Chapter (7):** JSON

  ***What is JSON? and, what JSON can do?***

# Chapter 1

# Introduction to XML

# XML Demo

- **<u>Demo (1):</u>**
  - Make a menu using string array its values written in the java file.
  - **The problem here is:**
    - If you want to change the name of a menu item or the number of menu items,
    - You will do that in the code so you have to recompile your java file.

# XML Demo (cont'd)

- **Demo (2):**
  - Make a dynamic menu and a dynamic button.
  - Set its position is dynamic also using string arrays read their values from a text file.
  - This program solves the problem of the 1st program, if you want to change the menu number, names, button names or position you do that simply in the file.

## XML Demo (cont’d)

- **The problem here is:**

  that button set position take only '4 case sensitive' values "North, South, East or West" so if you insert anything else the program will not run, will through exception.

- The **exception** tells that your code has a problem while the problem is in your text file.

## XML Demo (cont'd)

- **Demo (3):**
  - Make a dynamic menu and a dynamic button.
  - Set its position is dynamic using string arrays read their values from XML file
  - This program solves the problem of the 2nd program, if you insert a bad value in the XML document the program will not run, it will through exception.
  - But the exception thrown tells that your XML document has a problem.

## XML Introduction ?

- Stands for e**X**tensible **M**arkup **L**anguage.
- W3C Initiative.
- Created to provide a standard format for computer documents that is used as :
  - Websites documents
  - Electronic data interchange documents
    - (Example: Configuration file)

## What is XML?

- XML is a text file organized using tags to be:
  - **Standard based**
    - To be <u>language independent</u> and <u>platform independent</u>.
  - **Simple**
    - So it is '<u>human</u> and <u>machine</u> understandable'.
  - **Self describing** for the data it contains.
    - Because, it separates the content from its presentation
  - **Validated against strict rules** to
    - Make XML document optimized for computer manipulation
    - Be easy for data exchange

## What XML is? (Cont’d)

- XML, unlike HTML,

  Doesn’t have a fixed or predefined set of tags,

  The designer of the XML document invent his own set of tags.

# Chapter 2

## Creating Markup with XML

## Introduction

- **XML:**
  - Technology for creating markup languages
  - Enables authors to describe data of any type
  - Allows creating new tags
    - HTML limits authors to fixed tag set
  - Commonly stored in text files
    - Extension **`.xml`**
  - Example: [**`intro.xml`**]

# XML Example

```
<?xml version="1.0"?>

<!--              intro.xml              -->
<!-- Simple introduction to XML markup -->

<MyMessage>

    <message id='11'> Welcome to XML </message>

</MyMessage>
```

Document begins with declaration that specifies XML version 1.0

comments

Element **message** is child element of root element **MyMessage**

# 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.
  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.

## Parsers

- **XML parser:**
  - Processes XML document:
    - Reads XML document.
    - Checks syntax.
    - Reports errors (if any).
  - Example:
    - Internet browser
    - XML Editors.
    - Built in component Java JDK.

## Characters

- **Characters:**
  - ASCII characters:
    - Letters of English alphabet
    - Digits (**`0-9`**)
    - Punctuation characters, such as **`!`** , **`-`** and **`?`**
    - Carriage returns “\r” .
    - Line feeds “\n”.
  - Unicode characters:
    - Enables computers to process characters for several languages.

## Entity References & Built-in Entities

- XML **Reserved** characters:
  - Ampersand (`&`)
  - Left-angle bracket (`<`)
  - Right-angle bracket (`>`)
  - Apostrophe (`'`)
  - Double quote (`"`)

- But How to make this sequence
  - “ `5 < x && y > 6`” in element `message ?!!!`

## Entity References and Built-in Entities (Cont'd)

- **Entity references (Example)**
  - Allow to use XML-reserved characters
    - Begin with ampersand (`&`) and end with semicolon (`;`)

- **Built-in entities:**

| | |
|---|---|
| (&) → (&amp;amp;) | (<) → (&amp;lt;) |
| (>) → (&amp;gt;) | (’) → (&amp;apos;) |
| (”) → (&amp;quot;) | |

- So

  `“ 5 < x && y > 6”` → written as

  ```
  5 &lt; x &amp;&amp; y &gt; 6
  ```

## Attributes

- **Attributes:**
  - Elements may have associated attributes:
  - Placed within element's start tag:
  - Values are enclosed in quotes (single or double):
    - Element **`Father`** contains
      - attribute ***name***, which has value ***"Ahmad"***

```
<Father name = "Ahmad">
    <Child> Mohamad </Child>
</Father>
```

## Processing instruction

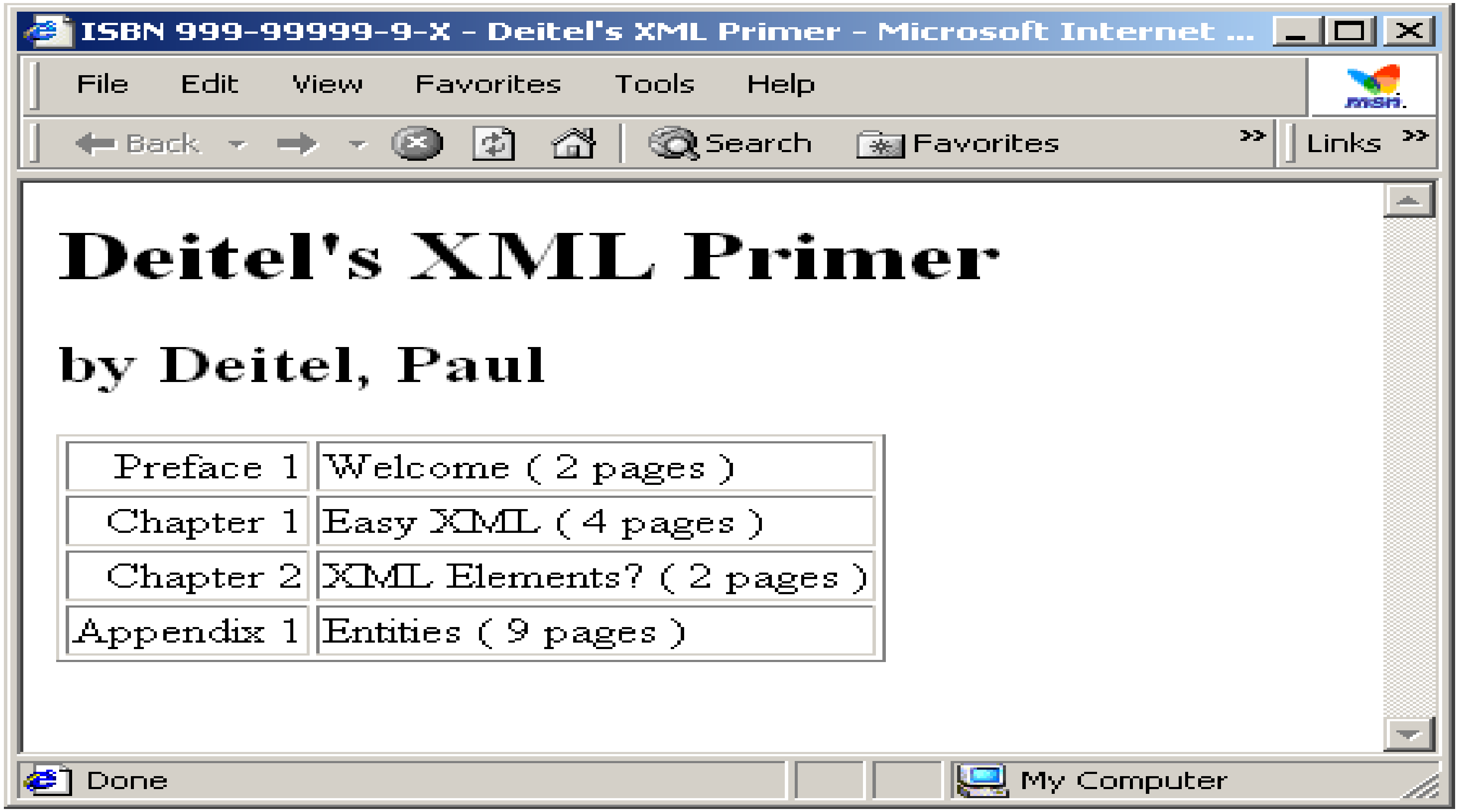
- **Processing instruction (PI)**
  - Passed to application using XML document.
  - Provides application-specific document information.
  - Delimited by **`<?`** and **`?>`**

# PI Example

```
<?xml version = "1.0"?>

<!-- Fig. 5.5 : usage.xml                  -->
<!-- Usage of elements and attributes -->

<?xml:stylesheet type = "text/xsl" href = "usage.xsl"?>

<book isbn = "999-99999-9-X">
   <title>Deitel&amp;s XML Primer</title>

   <author>
      <firstName>Paul</firstName>
      <lastName>Deitel</lastName>
   </author>

   <chapters>
      <preface num = "1" pages = "2">Welcome</preface>
      <chapter num = "1" pages = "4">Easy XML</chapter>
      <chapter num = "2" pages = "2">XML Elements?</chapter>
      <appendix num = "1" pages = "9">Entities</appendix>
   </chapters>

   <media type = "CD"/>
</book>
```

Stylesheet (discussed in Chapter 7)

# Present XML as HTML page Using XSL



**Deitel's XML Primer**

**by Deitel, Paul**

| | |
|---|---|
| Preface 1 | Welcome ( 2 pages ) |
| Chapter 1 | Easy XML ( 4 pages ) |
| Chapter 2 | XML Elements? ( 2 pages ) |
| Appendix 1 | Entities ( 9 pages ) |

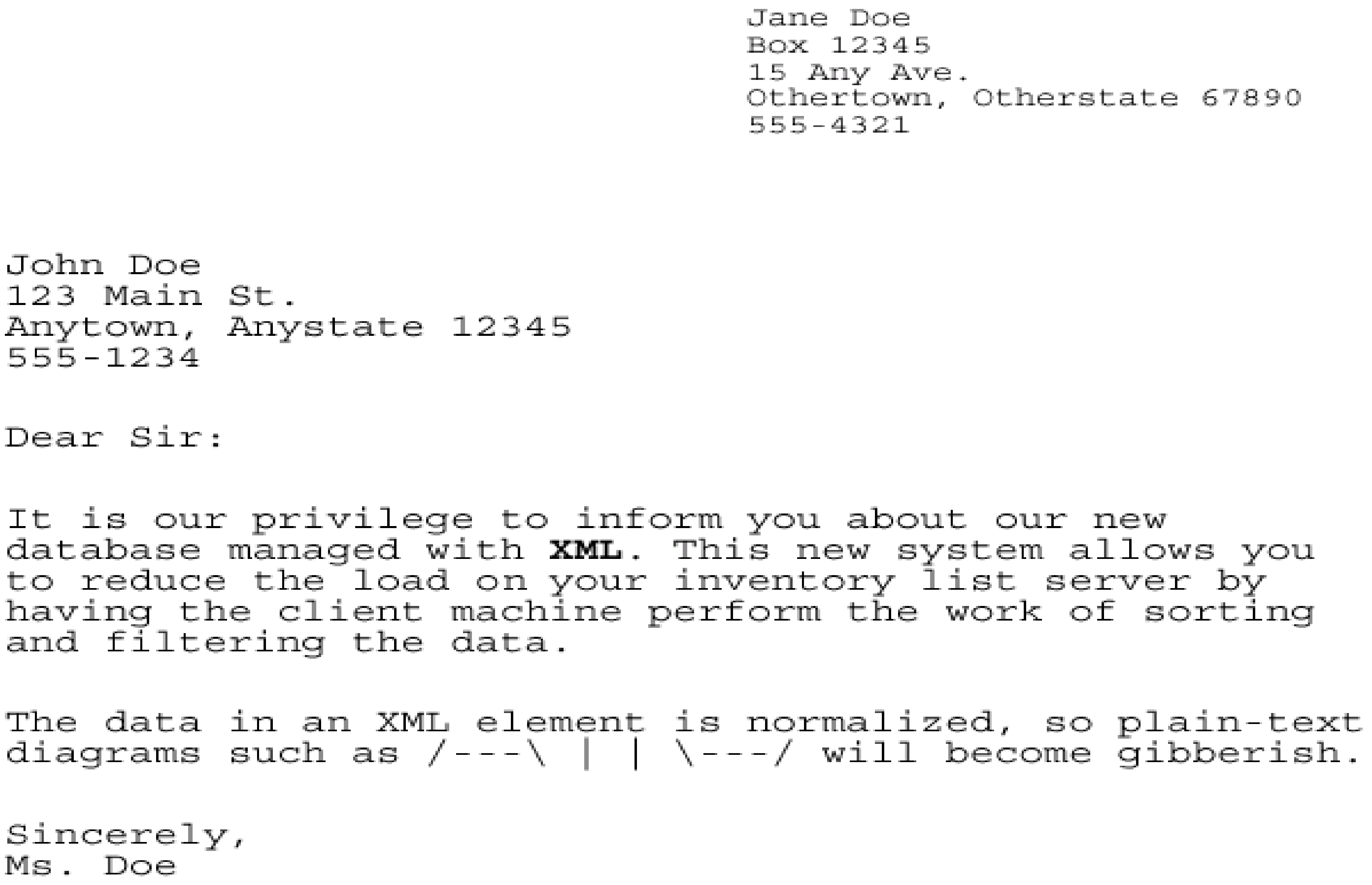
# PI Example 2

```xml
<?xml version = "1.0"?>

<!-- Fig. 5.6: letter.xml                     -->
<!-- Business letter formatted with XML -->

<letter>

   <contact type = "from">
      <name>Jane Doe</name>
      <address1>Box 12345</address1>
      <address2>15 Any Ave.</address2>
      <city>Othertown</city>
      <state>Otherstate</state>
      <zip>67890</zip>
      <phone>555-4321</phone>
      <flag gender = "F"/>
   </contact>

   <contact type = "to">
      <name>Jane Doe</name>
      <address1>123 Main St.</address1>
      <address2></address2>
      <city>Anytown</city>
      <state>Anystate</state>
      <zip>12345</zip>
      <phone>555-1234</phone>
      <flag gender = "M"/>
   </contact>
```

# PI Example 2 (cont'd)

```
   <salutation>Dear Sir:</salutation>

   <paragraph>It is our privilege to inform you about our new
      database managed with <bold>XML</bold>. This new system
      allows you to reduce the load on your inventory list
      server by having the client machine perform the work of
      sorting and filtering the data.</paragraph>

   <paragraph>The data in an XML element is normalized, so
      plain-text diagrams such as
         /---\
         |   |
         \---/
      will become gibberish.</paragraph>

   <closing>Sincerely</closing>
   <signature>Ms. Doe</signature>

</letter>
```

# Present XML As PDF Document Using XSL-FO.



```
                                                  Jane Doe
                                                  Box 12345
                                                  15 Any Ave.
                                                  Othertown, Otherstate 67890
                                                  555-4321



John Doe
123 Main St.
Anytown, Anystate 12345
555-1234

Dear Sir:

It is our privilege to inform you about our new
database managed with XML. This new system allows you
to reduce the load on your inventory list server by
having the client machine perform the work of sorting
and filtering the data.

The data in an XML element is normalized, so plain-text
diagrams such as /---\ | | \---/ will become gibberish.

Sincerely,
Ms. Doe
```

## CDATA Sections

- **`CDATA`** sections:
  - 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 `]]>`

# CDATA Example

```
<?xml version = "1.0"?>

<!-- Fig. 5.7 : cdata.xml                 -->
<!-- CDATA section containing C++ code    -->

<book title = "C++ How to Program" edition = "3">

   <sample>
         // C++ comment
         if ( this-&gt;getX() &lt; 5 &amp;&amp; value[ 0 ] != 3 )
            cerr &lt;&lt; this-&gt;displayError();
   </sample>

   <sample>
      <![CDATA[

         // C++ comment
         if ( this->getX() < 5 && value[ 0 ] != 3 )
            cerr << this->displayError();
      ]]>
   </sample>

   C++ How to Program by Deitel &amp; Deitel
</book>
```

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

XML does not process **CDATA** section

Note the simplicity offered by **CDATA** section

## Lab Exercise

## 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

# Chapter 3

# Document Type Definition (DTD)

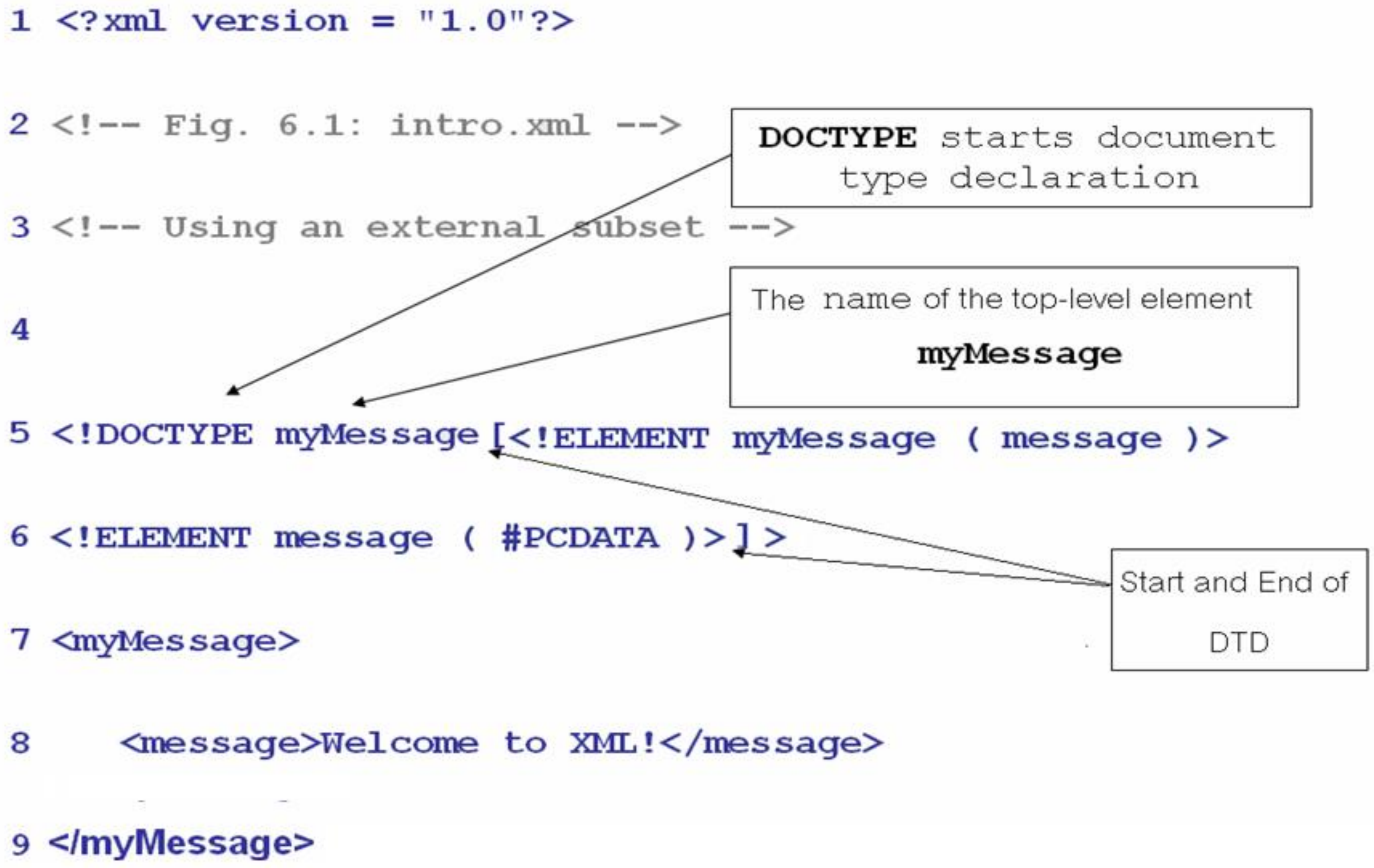
## Introduction

- **DTD:** Define structure of XML document.
- It defines:
  - The elements that can or must appear
  - How often the elements can appear
  - How the elements can be nested
  - Allowable, required and default attributes.

## DTD Declaration

- **DTD Can be categorized as:**
  - **Internal subsets:**
    - Declarations inside document
    - Visible only within document in which it resides
  - **External subsets:**
    - Declarations outside document
    - Exist in different file
      - typically ending with `.dtd` extension

# Example of Internal Subsets



```
<?xml version = "1.0"?>

<!-- Fig. 6.1: intro.xml -->

<!-- Using an external subset -->


<!DOCTYPE myMessage [<!ELEMENT myMessage ( message )>

<!ELEMENT message ( #PCDATA )>]>

<myMessage>

   <message>Welcome to XML!</message>

</myMessage>
```

**DOCTYPE** starts document type declaration
The name of the top-level element **myMessage**
Start and End of DTD

## DTD Declaration cont’d

- **DTD internal subsets** :
  - We declare DTDs in XML documents using DOCTYPE Syntax.
  - This line links the XML file to the DTD subset.

Begin with **`<!DOCTYPE`**

DTD –instructions ---

Ends with **>**

# Example of DTD file

```
<!-- Fig. 6.2: intro.dtd   -->

<!-- External declarations -->

<!ELEMENT myMessage ( message )>

<!ELEMENT message ( #PCDATA )>
```

Declare element **myMessage**

**Element** myMessage **contains child element** message

Declare element **message**

**Element** message **contains parsable character data**

# Example of External Subsets

```
<?xml version = "1.0"?>

<!-- Fig. 6.1: intro.xml -->
<!-- Using an external subset -->

<!DOCTYPE myMessage SYSTEM "intro.dtd">

<myMessage>
   <message>Welcome to XML!</message>
</myMessage>
```

**DOCTYPE** starts document type declaration

The name of the top-level element **myMessage**

Keyword **SYSTEM** specifies external subset

URL, Absolute path, or Relative path

## Sequences

- ***Sequences:***
  - Specify order in which elements occur. (,) is a delimiter
  - **Example:**

DTD

```
<!ELEMENT Name(FName, LName)>
<!ELEMENT FName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
```

----------------------------------------

XML

```
<Name>
  <FName>Khaled</FName>
  <LName>Ghazaly</LName>
</Name>
```

## Pipe Characters

- *Choice:*
  - Specify choices. (|) is a delimiter.
  - **Example:**

DTD

```
<!ELEMENT Sport (football | baseball)>
<!ELEMENT football (#PCDATA)>
<!ELEMENT baseball (#PCDATA)>
```

----------------------------------------

XML

```
<Sport>
  <football>
        Brazil
 </football>
</Sport>
```

```
<Sport>
  <baseball>
        Brazil
 </baseball>
</Sport>
```

## Occurrence Indicators

- ***Occurrence indicators:***
  - Plus sign (**+**) indicates **one** or **more** occurrences

    ```
    <!ELEMENT Book ( chapters+ )>
    ```

  - Asterisk (*****) indicates **zero** or **more** occurrences

    ```
    <!ELEMENT library ( book* )>
    ```

  - Question mark (**?**) indicates **zero** or **one** occurrences

    ```
    <!ELEMENT seat ( person? )>
    ```

## Element Type Declarations

- ***Simple declarations:***
  - Declare elements in XML documents

```
<!ELEMENT elementName (Content-Model)>
```

  - Content-Model can be:

1. **#PCDATA**: elements contains string content only
2. Another child **elements**
3. **Empty**
4. **ANY**
5. **Mixed**

## EMPTY, Mixed and ANY

3. **EMPTY:**
   - Elements do not contain character data
   - Elements do not contain child elements

**DTD**

```
<!ELEMENT Paragraph (MyLineBreak)>
<!ELEMENT MyLineBreak EMPTY>
```

   - Markup for:

**XML**

```
<Paragraph>
      <MyLineBreak/>
</Paragraph>
```

## EMPTY, Mixed and ANY (cont’d)

4. **ANY** “NOT Recommended”

   – Can contain any content:
     - `#PCDATA`,
     - Elements,
     - Combination of `#PCDATA` and Elements, or
     - empty element.

```
<!ELEMENT MyCustomElement ANY>
```

   – Commonly used in early DTD-development stages.

## EMPTY, Mixed and ANY (cont'd)

5. **Mixed** "NOT Recommended"

– Combination of elements and #**PCDATA**

DTD

```
<!ELEMENT myMessage ( #PCDATA | message )*>
```

– Markup for **myMessage:**

XML

```
<myMessage> Here is some text, some
     <message> other text </message> and
     <message> even more text </message>
</myMessage>
```

# Example: Mixed Element

```
<?xml version = "1.0" standalone = "yes"?>

<!-- Fig. 6.5 : mixed.xml         -->
<!-- Mixed content type elements -->

<!DOCTYPE format [
   <!ELEMENT format ( #PCDATA | bold | italic )*>
   <!ELEMENT bold ( #PCDATA )>
   <!ELEMENT italic ( #PCDATA )>
                ]>

<format>
   This is a simple formatted sentence.
   <bold>I have tried bold.</bold>
   <italic>I have tried italic.</italic>
   Now what?
</format>
```

Specify DTD as internal subset

Declare format as mixed content element

Elements bold and italic have PCDATA only for content specification

Element format adheres to structure in DTD

## Attribute Declarations

- ***Attribute declaration:***
  - Specifies element’s attribute list.
  - **General form:**

```
<!ATTLIST elementName
        AttributeName AttributeTypes AttributeBehavior
        AttributeName AttributeTypes AttributeBehavior
        AttributeName AttributeTypes AttributeBehavior
        . .…. . .
        AttributeName AttributeTypes AttributeBehavior >
```

# Example

```
<?xml version = "1.0"?>

<!-- Fig. 6.7: intro2.xml -->
<!-- Declaring attributes -->

<!DOCTYPE myMessage [
   <!ELEMENT myMessage ( message )>
   <!ELEMENT message ( #PCDATA )>
   <!ATTLIST message id CDATA #REQUIRED>
]>

<myMessage>

   <message id = "445">
      Welcome to XML!
   </message>
```

Specify DTD as internal subset

Declare element `myMessage` with child element `message`

Declare that attribute `id` contain required `CDATA`

## Attributes Behavior

Different Attribute-Behavior :

- Mandatory represented by:
  1. #REQUIRED

- Optional represented by:
  1. Immediate "default" Values
  2. #IMPLIED
  3. #FIXED

## Mandatory : REQUIRED

1. ***#REQUIRED:***

   – Attribute must appear in element.

   – Document is not valid if attribute is missing.

## Mandatory Example : REQUIRED

**(#REQUIRED)**

- ***In DTD:***

```
<!DOCTYPE Document[
  <!ELEMENT Document (Customer)*>
  <!ELEMENT Customer (NAME,DATE,ORDERS)>
            ...
  <!ATTLIST Customer Credit CDATA #REQUIRED> ]>
```

- ***In XML:***

```
<Document>
   <Customer Credit = "50"> ... </Customer> -→ Valid
   <Customer> ... </Customer> ----------------→ Invalid
</Document>
```

## Optional: Immediate Values

1. ***Immediate Values:***

   – It is as a default value If the attribute’s value is not present.

   – It is a simple text value, enclosed in quotes.

# Optional Example: Immediate Values

## (Immediate "or" Default Value)

- *In DTD:*

```
<!DOCTYPE Document[
    <!ELEMENT Document (Customer)*>
    <!ELEMENT Customer (NAME,DATE,ORDERS)>
           ...
    <!ATTLIST Customer Credit CDATA "0"> ]>
```

- *In XML:*

```
<Document>
    <Customer Credit = "50"> ... </Customer> → Valid
    <Customer> ... </Customer> --------------→ Valid
</Document>
```

## Optional : IMPLIED

2. ***#IMPLIED:***

   – Used when there is no default value for an attribute and you want to indicate that the author doesn't even have to use this attribute at all.
     – The programming layer decide the value of that attribute.

   – This keyword used when you want to allow the author to include an attribute but not require it.

## Optional Example: IMPLIED

**(#IMPLIED)**

- ***In DTD:***

```
<!DOCTYPE Document[
    <!ELEMENT Document (CUSTOMER)*>
    <!ELEMENT Customer (NAME,DATE,ORDERS)>
           ...
    <!ATTLIST Customer Credit CDATA #IMPLIED> ]>
```

- ***In XML:***

```
<Document>
  <Customer Credit ="$23.99"> ... </Customer> -→ Valid
  <Customer> ... </Customer>  ----------------→ Valid
</Document>
```

## Optional : FIXED

3. #FIXED:

   – They can appear in elements or not.

   – It must take attribute value.

   – Attribute value is constant

     • Attribute must always have that value.

     • Attribute value cannot differ in XML document.

# Optional Example: FIXED

## (#FIXED)

- ***In DTD:***

```
<!DOCTYPE Document [
  <!ELEMENT Document (Customer)*>
  <!ELEMENT Customer (NAME,DATE,ORDERS)>
          ...
  <!ATTLIST Customer LANGUAGE CDATA #FIXED "EN"> ]>
```

- ***In XML:***

```
<Document>
  <Customer> ... </Customer>  ----------------→ Valid
  <Customer LANGUAGE="EN"> . . . </Customer> → Valid
  <Customer LANGUAGE="AR"> . . . </Customer> → Invalid
</Document>
```

## Attribute Types

- **Attribute Types:**

1. Strings (CDATA)
   - No constraints on attribute values
     - Except for **disallowing <**, **>**, **&**, **'** and **"** characters
2. ID
3. IDREF
4. NMTOKEN
5. Enumerated

# Attribute Types (cont'd)

## 2. *ID:*

- The value is used to identify elements.
- ID value must begin with
  - a letter, underscore (_) or a colon (:)
- ID value is unique per document.
  - Mean no other ID type attribute for any element in the document can have the same value.
- You can't use the ***ID*** type with ***#FIXED*** attributes or **Immediate** values.
- Providing more than one ID attribute type **for an element** is a logical error but not DTD error.

## Attribute Types (cont'd)

3. *IDREF*

   – Points to elements with **ID** attribute.

   – **IDREF** hold the **ID** value of another element in the document.

- <u>**Note:**</u>

  **ID** and **IDREF** must have a declared <u>behavior</u> of #**IMPLIED** or #**REQUIRED**.

# Example

```
<?xml version = "1.0"?>

<!-- Fig. 6.8: IDExample.xml -->
<!-- Example for ID and IDREF values of attributes -->

<!DOCTYPE bookstore [
   <!ELEMENT bookstore ( shipping+, book+ )>
   <!ELEMENT shipping ( duration )>
   <!ATTLIST shipping shipID ID #REQUIRED>
   <!ELEMENT book ( #PCDATA )>
   <!ATTLIST book shippedBy IDREF #IMPLIED>
   <!ELEMENT duration ( #PCDATA )>
]>

<bookstore>
   <shipping shipID = "s1">
      <duration>2 to 4 days</duration>
   </shipping>

```

**Each** shipping **element has a unique identifier (**shipID**)**

**Attribute** shippedBy **points to shipping element by matching** shipID **attribute1**

## Example (cont'd)

```
    <shipping shipID = "s2">
       <duration>1 day</duration>
    </shipping>

    <book shippedBy = "s2">
       Java How to Program 3rd edition.
    </book>

    <book shippedBy = "s2">
       C How to Program 3rd edition.
    </book>

    <book shippedBy = "s1">
       C++ How to Program 3rd edition.
    </book>
</bookstore>
```

**Declare** `book` **elements with attribute** `shippedBy`

## Attribute Types (cont'd)

**4. NMTOKEN:**

- Tokenized attribute type
- "Name token"
- Value consists of letters, digits, periods, underscores, hyphens and colon characters
- It Just Take One word,
  - it can't contain spaces, comma.

## Example

*IN DTD:*

```
<!ELEMENT company (#PCDATA)>
<!ATTLIST company Address NMTOKEN #REQUIRED>
```

*In XML:*

```
<company Address = “241 ElAhram St"> not valid
<company Address = “241_ElAhram:St"> valid
```

## Attribute Types (cont'd)

### 5. *Enumerated:*

- Declare list of possible values for attribute

```
<!ATTLIST person gender ( M | F ) "F">
```

  - Attribute **`gender`** can have either value **`M`** or **`F`**
  - **`F`** is default value.

## ENTITY

***ENTITY***

– Entity is a place to store text data, like constant in JAVA.

– **General Entity declaration:**

```
<!ENTITY entity-name "entity-value">
```

## ENTITY

- **<u>Example:</u>**

*In DTD:*

```
<!ENTITY My_Address “241 El-Haram st. Giza”>
```

*In XML:*

```
<address>&My_Address;</address>
```

– Entity reference `&My_Address;` replaced by its value

```
<address>241 El-Haram st. Giza</address>
```

## ENTITY

- **External General Entity declaration:**

```
<!ENTITY My_Address SYSTEM “My_Addr.ent”>
```

  - **In file My_Addr.ent:**
    - `241 El-Haram st. – Giza`
  - Entity may be used as follows:

```
<useAnEntity>&My_Address;</useAnEntity>
```

  - Entity reference **&My_Address;** replaced by its value

```
<useAnEntity> 241 El-Haram st. – Giza</useAnEntity>
```

## Assignment

- Design a DTD of the configuration file for a library that you made.

- Note:
  - External DTD
  - In DTD:
    - Define elements with occurrence indicators
    - Define attribute with different types (CDATA, enumerated,…)
      with different behavior (required, optional)
    - Define external entity