HUMBER

XML and JavaScript
ITC5202

Week 2

Lecturer: Shahdad Shariatmadari

Winter 2022

Agenda

- XML Structure
  - Declaration
  - Elements
  - Attributes
  - Entities
- XML DTD
- Valid vs well-formed document

Introducing XML

- **XML** stands for **Extensible Markup Language**
  - It is markup language that can be extended and modified to match the needs of the document author and data being recorded
  - XML has some advantages in presenting structured content
  - Because it is extensible, XML can be used to create a wide variety of document types

XML Today

- XML was originally created to structure, store, and transport information
- XML has become the most common tool for data transmission among various applications
- All major databases can read and create XML files

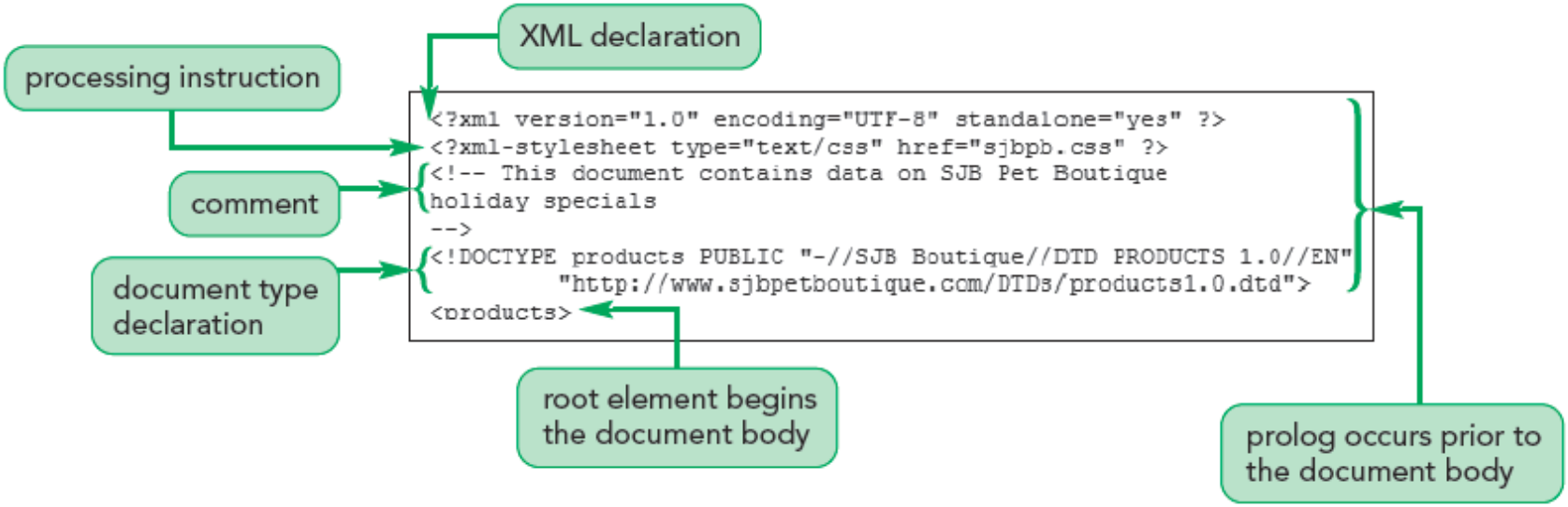
Example of an XML file

```
<?xml version="1.0" encoding="utf-8"?>
<myMessage>
        <subject> This is a welcome message
</subject>
        <date> Jan 31, 2018 </date>
</myMessage>
```

The Structure of an XML Document

- XML documents consist of three parts
  - The prolog
  - The document body
  - The epilog
- The prolog provides information about the document itself
  - XML declaration
  - Processing instructions
  - Comments lines
  - Document type declaration (DTD)

The Structure of an XML Document



The Structure of an XML Document

- The document body contains the document's content in a hierarchical tree structure
- The epilog is optional and contains any final comments or processing instructions

The XML Declaration

- The XML declaration is always the first part of the prolog in an XML document; it signals to the program reading the file that the document is written in XML, and it provides information about how that code is to be interpreted by the program
- The syntax is:

```
<?xml version="version number"
   encoding="encoding type" standalone="yes
   | no" ?>
```

- A sample declaration:

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

Inserting Comments

- Comments can appear anywhere in the prolog after the XML declaration
- Comments provide additional information about what the document will be used for and how it was created
- The syntax for comments is `<!-- comment -->`
- This is the same syntax for HTML comments

```
<!--
    This document contains data on SJB Pet
    Boutique holiday specials
    File name: sjbpet.xml
    Author: Patricia Dean        Date: 9/18/2017
-->
```

Working with Elements

- Elements are the basic building blocks of XML
- An element can have text content and child element content
- The content is stored between an opening tag and a closing tag, just as in HTML
- The syntax of an XML element with text:
  <element>content</element>
- Example:

```
<manufacturer>SJB Pet
   Boutique</manufacturer>
```

Working with Elements (continued)

- Element names are case sensitive
- Element names must begin with a letter or the underscore and cannot contain blank spaces
- The element's name in the closing tag must exactly match the name in the opening tag
- An empty element with a single tag: `<element />`
- An empty element with a pair of tags: `<element></element>`

Nesting Elements

- An element contained within another element
- Nested elements also called child elements
- Child elements must be enclosed within their parent elements
- Example:

```
<product>
    <productName>Dog Shirt Gift Basket</productName>
    <manufacturer>SJB Pet Boutique</manufacturer>
    <description>Something for every day of the
       week</description>
    <price>35.99</price>
    <productItems>1200, 1201, 1202, 1203,
       1204</productItems>
</product>
```

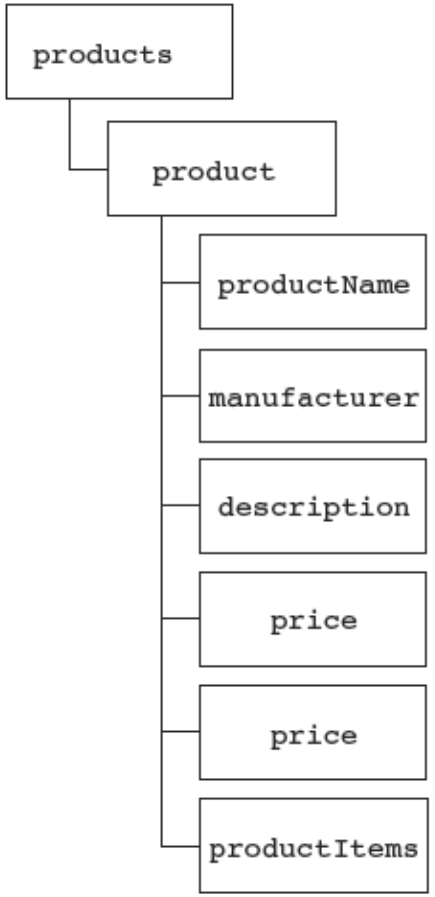
The Element Hierarchy (continued)

XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!--
   This document contains data on SJB Pet Boutique
   holiday specials

   Filename: sjbpet.xml
   Author:   Patricia Dean
   Date:     9/18/2017
-->
<products>
   <product>
      <productName>Dog Shirt Gift Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Something for every day of the week</description>
      <price>35.99</price>
      <price>26.79</price>
      <productItems>1200, 1201, 1202, 1203, 1204, 1205, 1206</productItems>
   </product>
</products>
<!-- generated by the finance department -->
```

Hierarchy tree structure



Working with Attributes

- An **attribute** describes a feature or characteristic of an element
- Every element can contain one or more attributes
- Attributes are text strings and must be placed in single or double quotes. The syntax is:

```
<element attribute="value"> … </element>
```

or

```
<element attribute="value" />
```

How to use Attribute

- There are many ways to design XML files, and the most common design is not to use attributes for data.
- Instead they are used to indicate some property about the data that's in the content of the tag.
- Another way to phrase this is to say that they are used for metadata.

Example

- <fileSize unit=“KB”> 34.55 </fileSize>
- <cost currency=“USD”> 125.75 </cost>

Activity

- How to design an XML file for the following business card?

Activity

- Assume that you have multiple business cards and you want to add them to the XML document that you design.
  - Do you need to modify your XML structure?
- Try to use attributes in your design.

Writing the Document Body

Figure 1-25

Activity

- Download Sample product data from BB->Module2.
- There are three prices for each product. Modify the XML file and add currency attribute to each price as follow
  - USD
  - CAD
  - EURO
- Modify the XML file
  - Add currency symbol to the price data
  - Add Trademark symbol to the manufacturer data

Working with Attributes

Figure 1-27

attribute name and value are defined in the opening tag of each element

```
<products>
   <product>
      <productName>Dog Shirt Gift Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Something for every day of the week</description>
      <price currency="USD">35.99</price>
      <price currency="EUR">26.79</price>
      <productItems>1200, 1201, 1202, 1203, 1204, 1205, 1206</productItems>
   </product>
   <product>
      <productName>Cat Curiosity Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Playtime morning, noon, and night</description>
      <price currency="USD">15.99</price>
      <price currency="EUR">11.90</price>
      <productItems>4430, 6500, 4434</productItems>
   </product>
   <product>
      <productName>Piggy Snuggle Basket</productName>
      <manufacturer>ACME</manufacturer>
      <price currency="USD">17.50</price>
      <price currency="EUR">13.03</price>
      <productItems>3230, 3232</productItems>
   </product>
   <product>
      <productName>Dog Snuggle Basket</productName>
      <manufacturer>ACME</manufacturer>
      <price currency="USD">14.25</price>
      <price currency="EUR">10.61</price>
      <productItems>3230, 3232, 3250</productItems>
   </product>
</products>
```

Using Character and Entity References

- Special characters, such as the € symbol, can be inserted into your XML document by using a **character reference**; the syntax is:

  `&#nnn;`

- Some symbols also can be identified using an **entity reference**; the syntax is:

  `&entity;`

Using Character and Entity References

- *nnn* is a character reference number or name from the ISO/IEC character set
- *entity* is the name assigned to the symbol
- **ISO/IEC character set** is an international numbering system for referencing characters from virtually any language
- Character references in XML are the same as in HTML

XML Supported Entity Names

- XML supports the following five built-in entities:
  - `&` for the & character
  - `<` for the < character
  - `>` for the > character
  - `&apos;` for the ‘ character
  - `&quot;` for the “ character
- When an XML parser encounters these entities, it can display the corresponding character symbol
- For the other special characters (like ©) you need to use **character reference**.

Using Character and Entity References

Symbol	Character Reference	Entity Reference	Description
>	&#62;	&gt;	Greater than
<	&#60;	<_	Less than
'	&#27;	&apos;	Apostrophe (single quote)
"	&#22;	&quot;	Double quote
&	&#38;	&amp;	Ampersand
©	&#169;	&copy;	Copyright
®	&#174;	&reg;	Registered trademark
TM	&#153;		Trademark
0	&#176;		Degree
£	&#163;		Pound
€	&#8364;	&euro;	Euro
¥	&#165;	&yen;	Yen

Using Character and Entity References

Figure 1-30


```
<products>
   <product>
      <productName>Dog Shirt Gift Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Something for every day of the week</description>
      <price currency="USD">$35.99</price>
      <price currency="EUR">&#8364;26.79</price>
      <productItems>1200, 1201, 1202, 1203, 1204, 1205, 1206</productItems>
   </product>
   <product>
      <productName>Cat Curiosity Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Playtime morning, noon, and night</description>
      <price currency="USD">$15.99</price>
      <price currency="EUR">&#8364;11.90</price>
      <productItems>4430, 6500, 4434</productItems>
   </product>
   <product>
      <productName>Piggy Snuggle Basket</productName>
      <manufacturer>ACME</manufacturer>
      <price currency="USD">$17.50</price>
      <price currency="EUR">&#8364;13.03</price>
      <productItems>3230, 3232</productItems>
   </product>
   <product>
      <productName>Dog Snuggle Basket</productName>
      <manufacturer>ACME</manufacturer>
      <price currency="USD">$14.25</price>
      <price currency="EUR">&#8364;10.61</price>
      <productItems>3230, 3232, 3250</productItems>
   </product>
</products>
```

Activity

- Add the following message to each product in the **product.xml**

```
<message>
  For further information, please visit <our website> or call us on #905-111-2222 & +1-800-111-2222
</message>
```

- Have you get any error? Why?

CDATA or Character Data

- Instead of modifying data in the previous activity in order to solve the problem, we can keep the data as it is in CDATA block.
- **Character data** is not processed, but instead is treated as pure data content

Creating a CDATA Section

- A CDATA section is a block of text that XML treats as character data only
- The syntax to create a CDATA section is:

```
<![CDATA[
   character data
]]>
```

- A CDATA section may contain most markup characters, such as <, >, and &

CDATA Example

Processing instructions

- Processing instructions are a way for XML content to deliver special instructions to the XML parser.
  - `<?targetName  instruction ?>`
- An example of why you might use a processing instruction is maybe you have an application that has multiple spell-checking modes and a particular document might want to indicate to your app that a particular language should be used when a document is being spell-checked.
  - `<?SpellCheckMode mode="en-GB" ?>`

Types of XML Content

- We learned about the following types of XML content

XML Document Declaration	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
Elements and Attributes	<element attribute="value">
Comments	<!-- This is an XML comment -->
Character Data	<![CDATA[This is unparsed text & data]]>
Processing Instructions	<?SpellCheckMode mode="us-english"?>
Entity References	Character (<) and General (&copyright;)

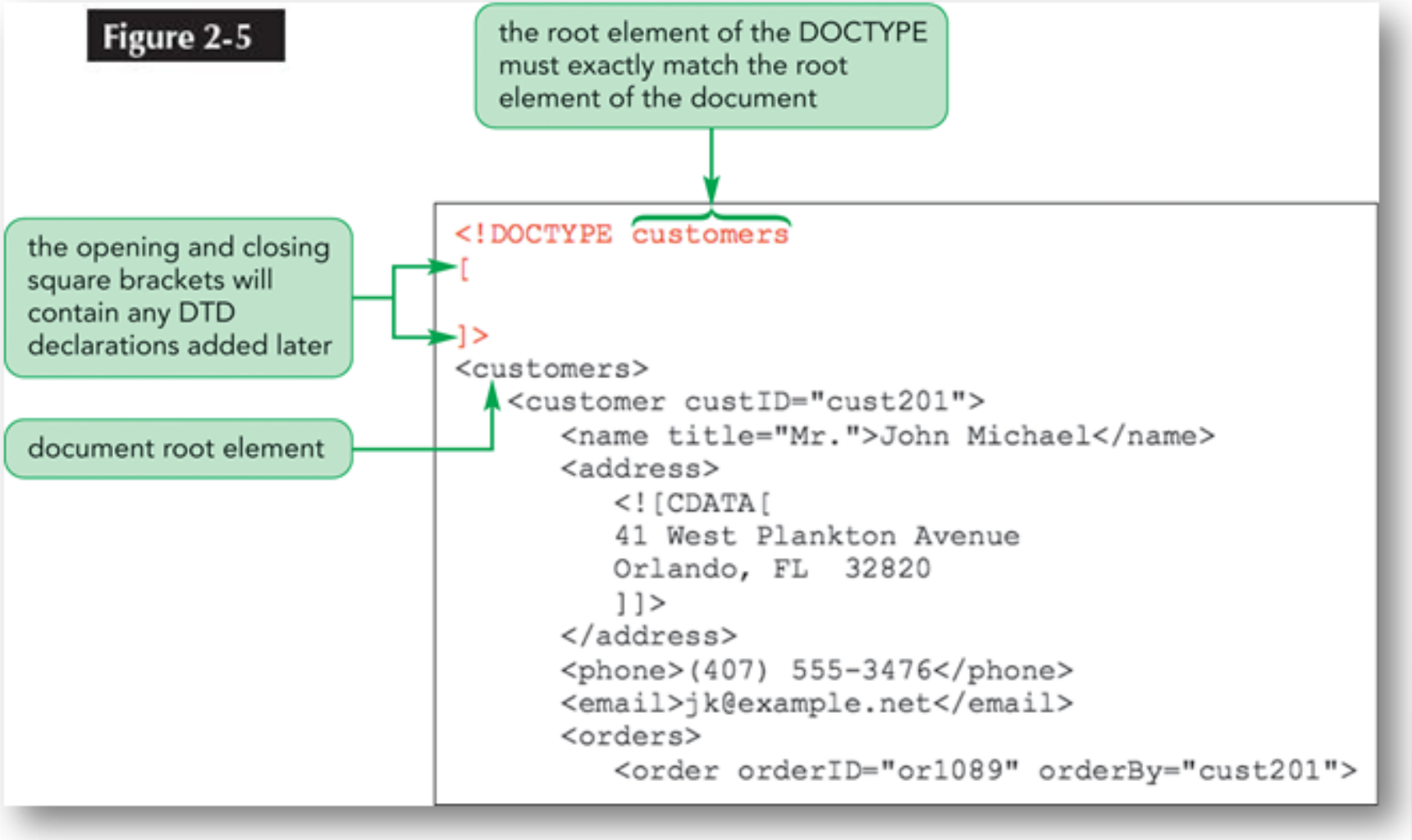
The Document Type Definition

- Document Type Definitions (DTDs) offer a way of specifying further rules that help in the interpretation of documents and their structure
- DTD includes the following declarations:
  - Element Type Declarations
  - Attribute List Declarations
  - Entity Declarations

Declaring a DTD

- The DTD(document type declaration) always begins with <!DOCTYPE, followed by some whitespace as follow:
  - <!DOCTYPE *root* [declaring elements and attributes]>
  - <!DOCTYPE *root* SYSTEM "*uri*">
- Each XML document can have only one DOCTYPE

Writing the Document Type Declaration



DTD declaration

- In addition to the different types of DTDs and their basic function, DTD declarations can be broken down into
  - Element Type Declarations
  - Attribute List Declarations
  - Entity Declarations

DTD main Skeleton

```
<!DOCTYPE rootElement [
      <!ELEMENT mainElement(child1,child2)>
      <!ELEMENT child1 (#PCDATA)>
      <!ELEMENT child2 (#PCDATA)>
]>
```

ELEMENT DECLARATION

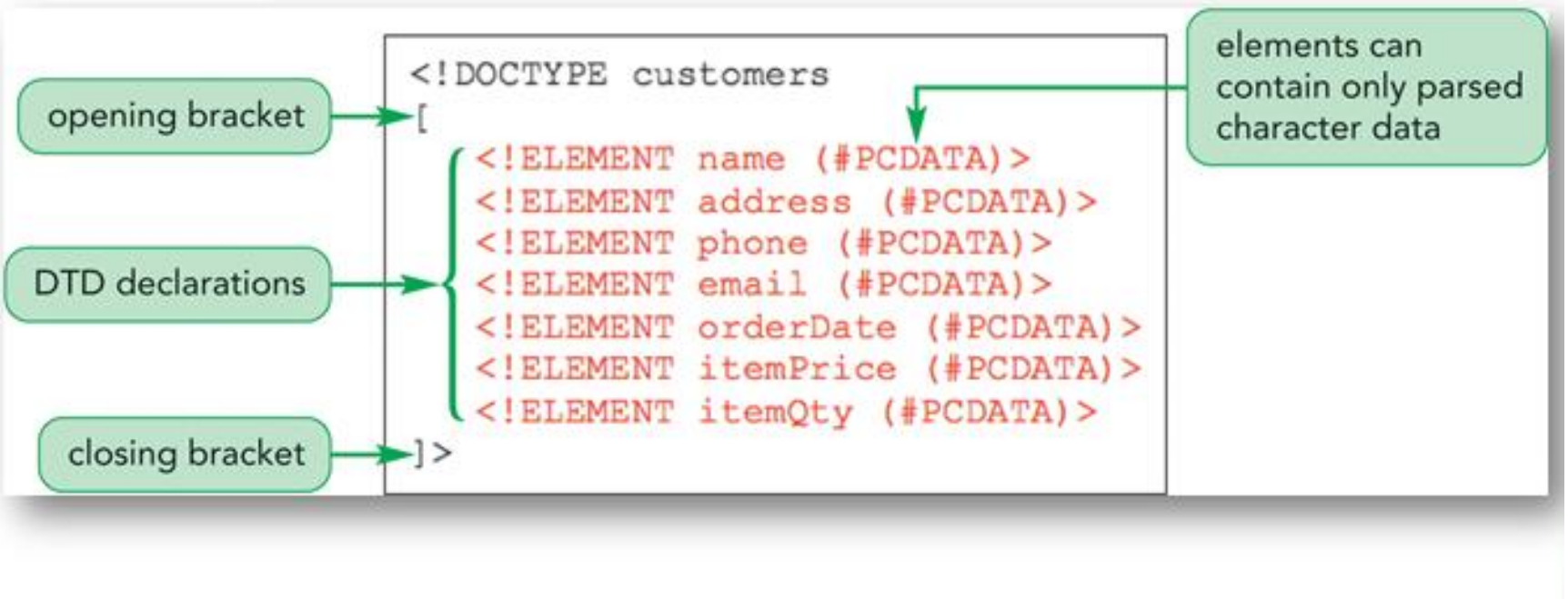
Declaring Document Elements

- Every element must be declared in the DTD
- An **element type declaration**, specifies an element's name and indicates what content the element can contain
- The syntax of an element declaration:
  `<!ELEMENT element content-model>`

Types of Element Content

- The `content-model` specifies what type of content the element contains:
  - #PCDATA: The element can contain only parsed character data
    - The syntax is: `<!ELEMENT element (#PCDATA)>`
    - `Example : <!ELEMENT name (#PCDATA)>`

Element Declaration Example



Types of `Content-Model`
`<!ELEMENT element content-model>`

- The `content-model` specifies what type of content the element contains:
  - `ANY`: The element can store any type of content or no content at all
  - `EMPTY`: The element cannot store any content
  - `#PCDATA`: The element can contain only parsed character data
  - `Sequence`: The element can contain only child elements
  - `#PCDATA with sequence`: The element can store both parsed character data and child elements

Working with Child Elements

- The syntax for declaring an element that contains only child elements is:
  `<!ELEMENT element (children)>`
- *element* is the parent element and *children* is a listing of its child elements
- Example: `<!ELEMENT customer (phone)>` indicates that the `customer` element can contain only a single child element named `phone`

Working with Child Elements

- A **sequence** is a list of elements that follow a defined order; the syntax is:
  `<!ELEMENT element (child1, child2, …)>`
- *child1*, *child2*, and so on, represents the sequence of child elements within the parent
- The order of the child elements in an XML document must match the order defined in the element declaration
- Example:

```
<!ELEMENT customer (name, phone,
email)>
```

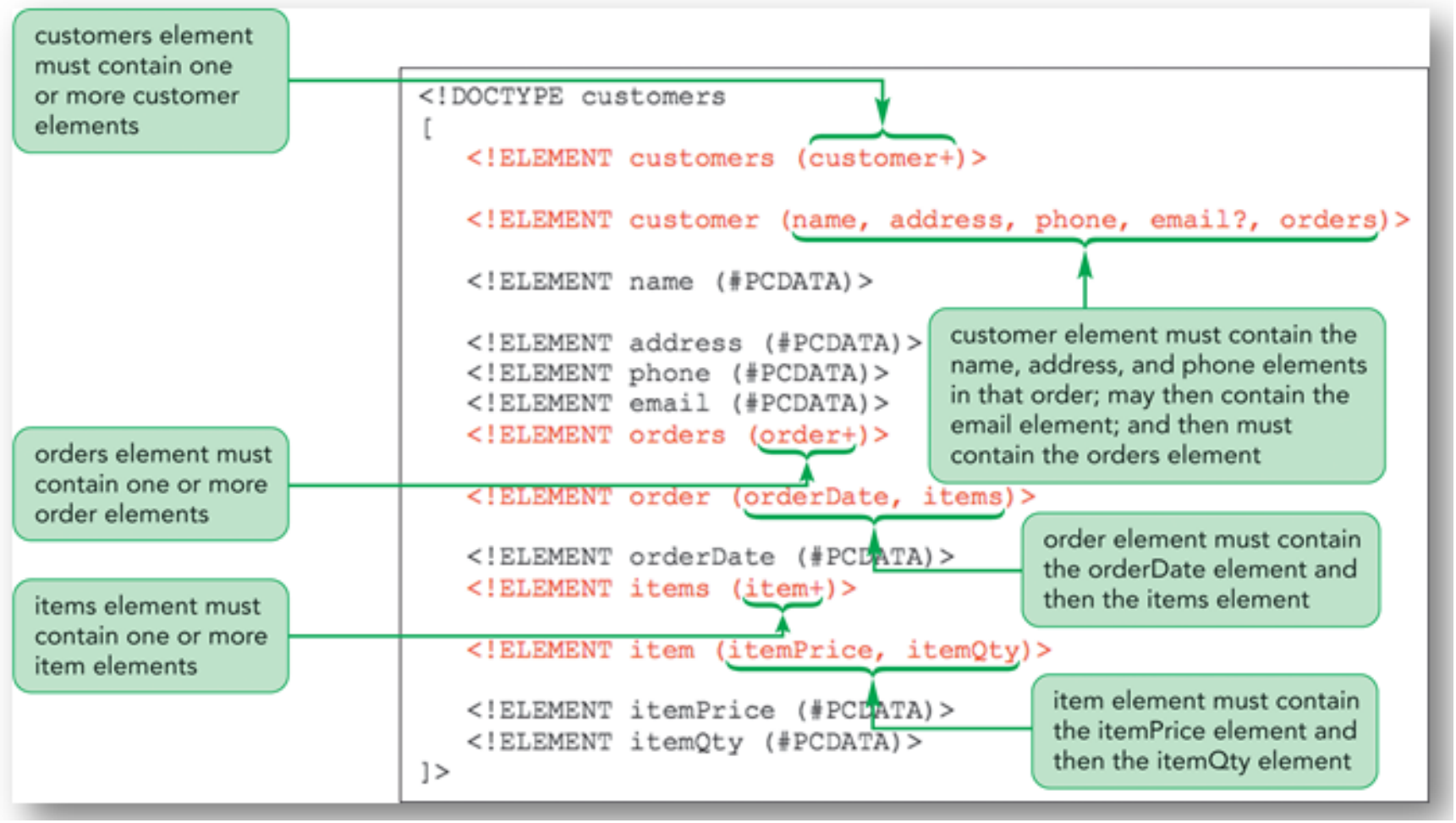
Cardinality

- An element's *cardinality* defines how many times it will appear within a content model
  - `<!ELEMENT customers (customer+)>`
- There are three modifying symbols:
  - A question mark (?)—indicates that an element occurs zero times or one time
  - A plus sign (+)—indicates that an element occurs at least once
  - An asterisk (*)—indicates that an element occurs zero times or more

Cardinality example

- Example:
  - `<!ELEMENT customers (customer+)>`
  - This allows the document to contain one or more customer elements to be placed within the customer element
- The three modifying symbols can also modify entire element sequences or choices, for example:
  - `<!ELEMENT order (orderDate, items)+>`

Declaration Example



Specifying an Element Choice

- The element declaration can define a **choice** of possible elements; the syntax is:
  `<!ELEMENT element (child1 | child2 | …)>`
  - Example: `<!ELEMENT customer (name | company)>`
- This allows the customer element to contain either the name element or the company element

Activity

- Do some research and explain
- What is well-formed XML document?
- What is a valid XML document?
- Is Product.XML a well-formed XML document?
- Is Product.XML a VALID XML document?
  - Use https://www.xmlvalidation.com/ to validate it.

Activity

- Develop a DTD for the Product.xml (the original file without adding currency and message)