Extensible Markup Stylesheet Transformation (XSLT)

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Dept. of Computer Engineering
Khon Kaen University

Overview

- Terms: XSL, XSLT, XSL-FO
- Value of Transformation
- XSLT Operational Model
- Apache Xalan
- Template Rules
- XSLT Stylesheet Language
- Programming API

XSL – The Style Sheet of XML

- XML does not use predefined tags (we can use any tags we want)
- <table> could mean an HTML table, a piece of furniture, or something else
- XSL: something in addition to the XML document that describes how the document should be displayed

What is XSL?

- eXtensible Stylesheet Language
- A language for expressing stylesheets
- Make up of two parts
  - XSL Transformation (XSLT)
  - XSL Formatting Objects (XSL-FO)

Transformation

- Transforming XML document into
  - Another XML document
    - XHTML
    - WML
  - HTML document
  - Text
- XSLT
  - W3C standard for XML transformation

Two Viewpoints of XML

- Presentation Oriented Publishing (POP)
  - Useful for Browsers and Editors
  - Usually used for data that will be consumed by Humans
- Message Oriented Middleware (MOM)
  - Useful for Machine-to-Machine data exchange
  - Business-to-Business communication

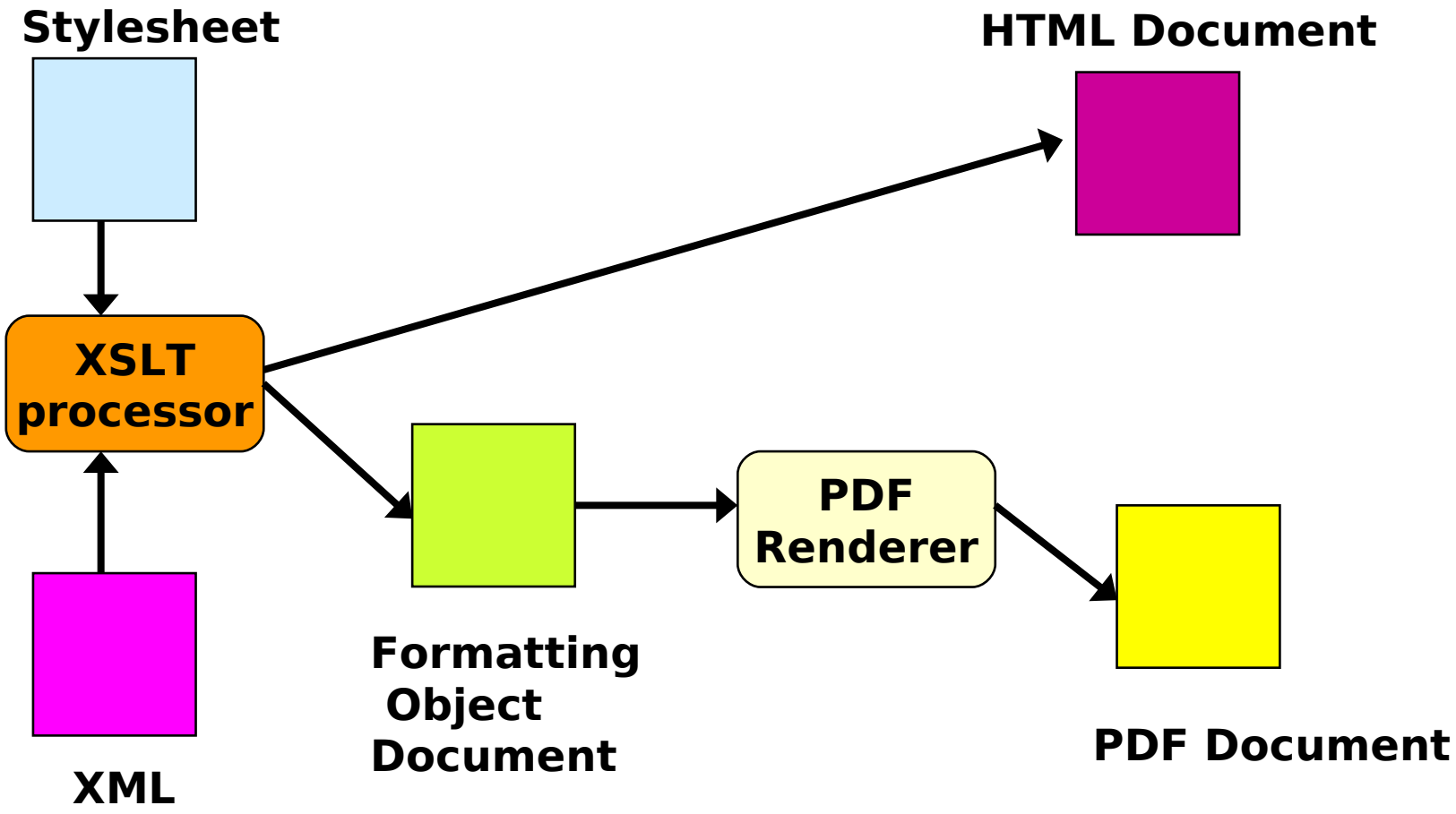
Importance of Transformation

- XSLT is incredibly useful in
  - Transforming data into a viewable format in a browser
  - Transforming business data between content models

XSLT in POP

- XML document separates content from presentation
- Transformations can be used to style (render, present) XML documents
- A common styling technique presents XML in HTML format

XSLT in POP Example

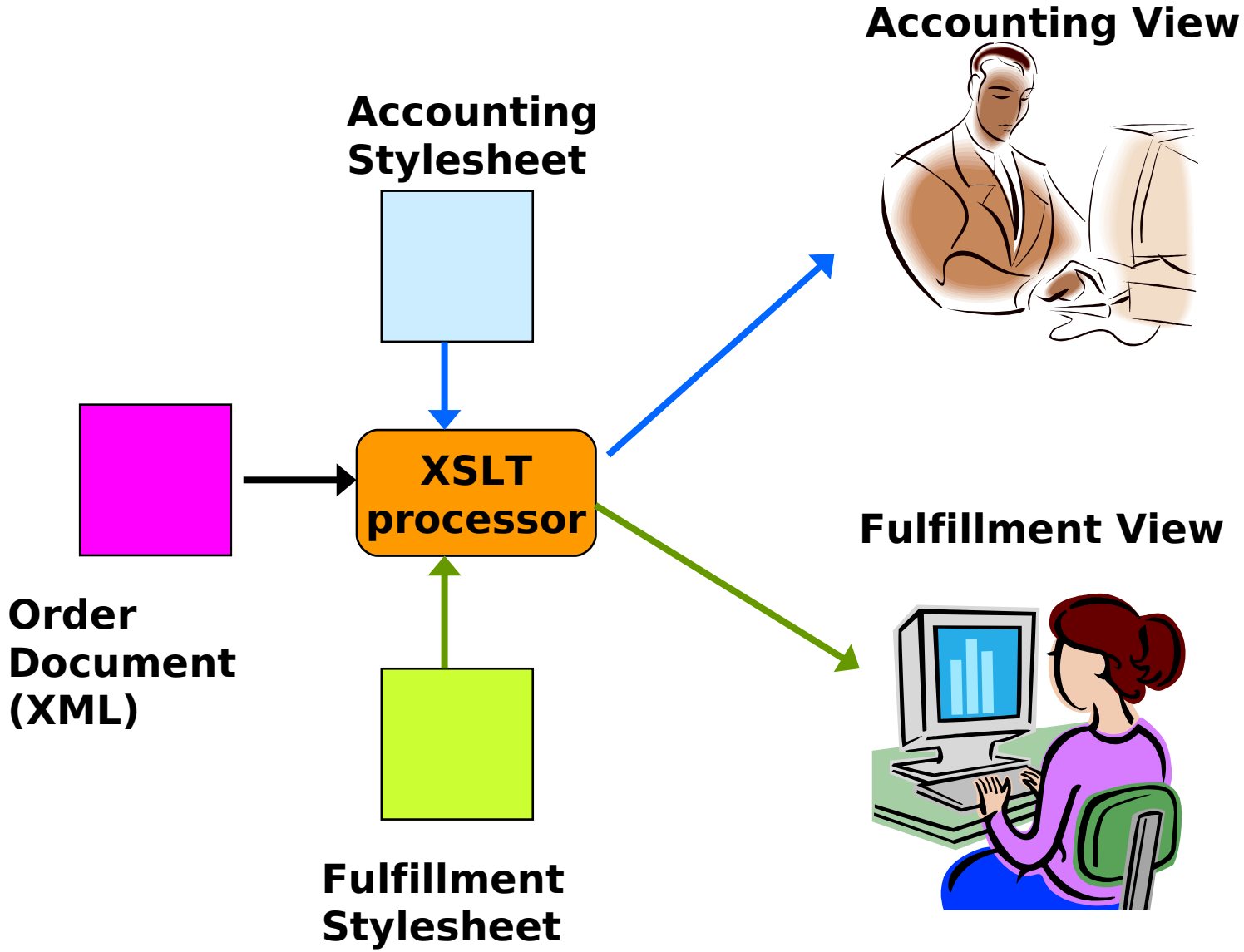


XSLT in MOM

- Important for eCommerce, B2B/EDI,

  and dynamic content generation
  - Different content model
  - Different structural relationship
  - Different vocabularies

XSLT in MOM Example



What is XSLT? (1/2)

- XSLT is a transformation language
- XSLT is designed as a templating language
- An XSLT stylesheet describes how documents in one format are converted to documents in another format
- Both input and output documents are represented by the XPath data model

What is XSLT? (2/2)

- XPath expression select nodes from the input document for further processing
- Templates containing XSLT instructions are applied to the selected nodes to generate new nodes that are added to the output document
- XSLT is based on the notion of templates

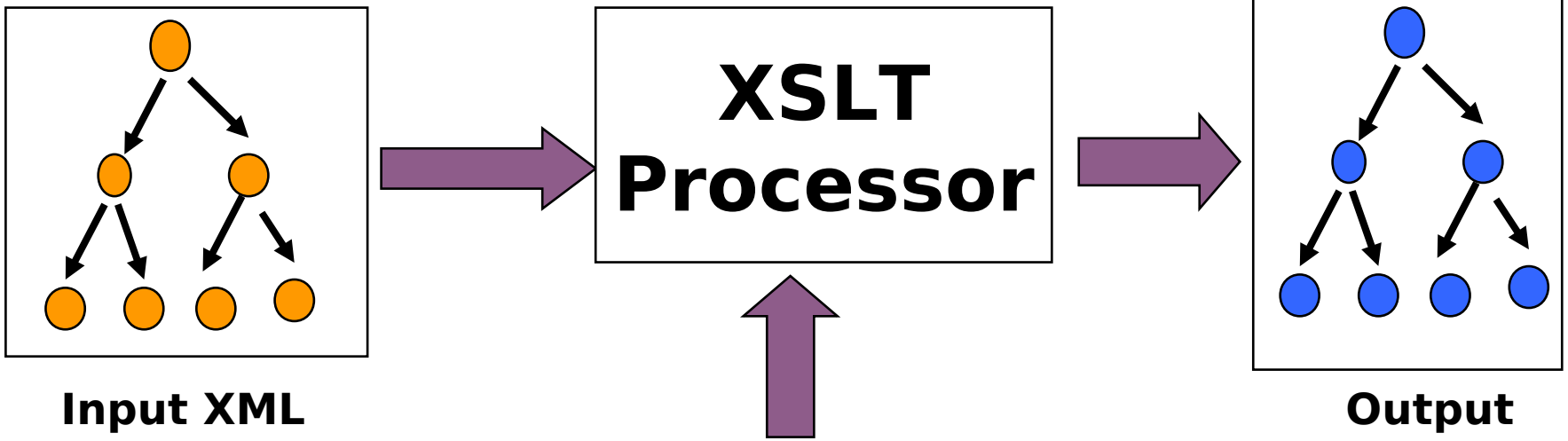
How Does XSLT Work? (1/2)

- XSLT transforms an XML source tree into an XML result tree
- XSLT uses XPath to define parts of the source document that match one or more predefined templates
- XSLT is rule-based

How Does XSLT Work? (2/2)

- When a match is found, XSLT engine will transform the matching part of the source document into the result document
- The parts of the source document that do not match a template will end up unmodified in the result document

XSLT Operational Model



Input XML

Output

XML
XHTML
HTML
WML
text
…

```
…
<xsl:template match="cd/title">
   <h2>
         <xsl:value-of="."/>
   </h2>
</xsl:template>
…
```

XSL Stylesheet

XSLT Processor

- Piece of software
  - Reads an XSLT stylesheet and input XML document
  - Converts the input document into an output document
    - According to the instruction given in the stylesheet
- Called stylesheet processor sometimes

Examples of XSLT Processor

- Built-in within a browser
  - Microsoft IE
  - Mozilla Firefox
- Built-in within a web development framework
  - Apache Cocoon
- Standalone
  - Michael Kay’s SAXON
  - Apache Xalan

Example: An Input XML Document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
   href="catalog.xsl"?>
<catalog>
   <cd country="UK">
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <price>9.90</price>
   </cd>
   <cd country="USA">
        <title>Greatest Hits</title>
        <artist>Dolly Parton</artist>
        <price>10.90</price>
   </cd>
</catalog>
```

Example: An Input XSL File

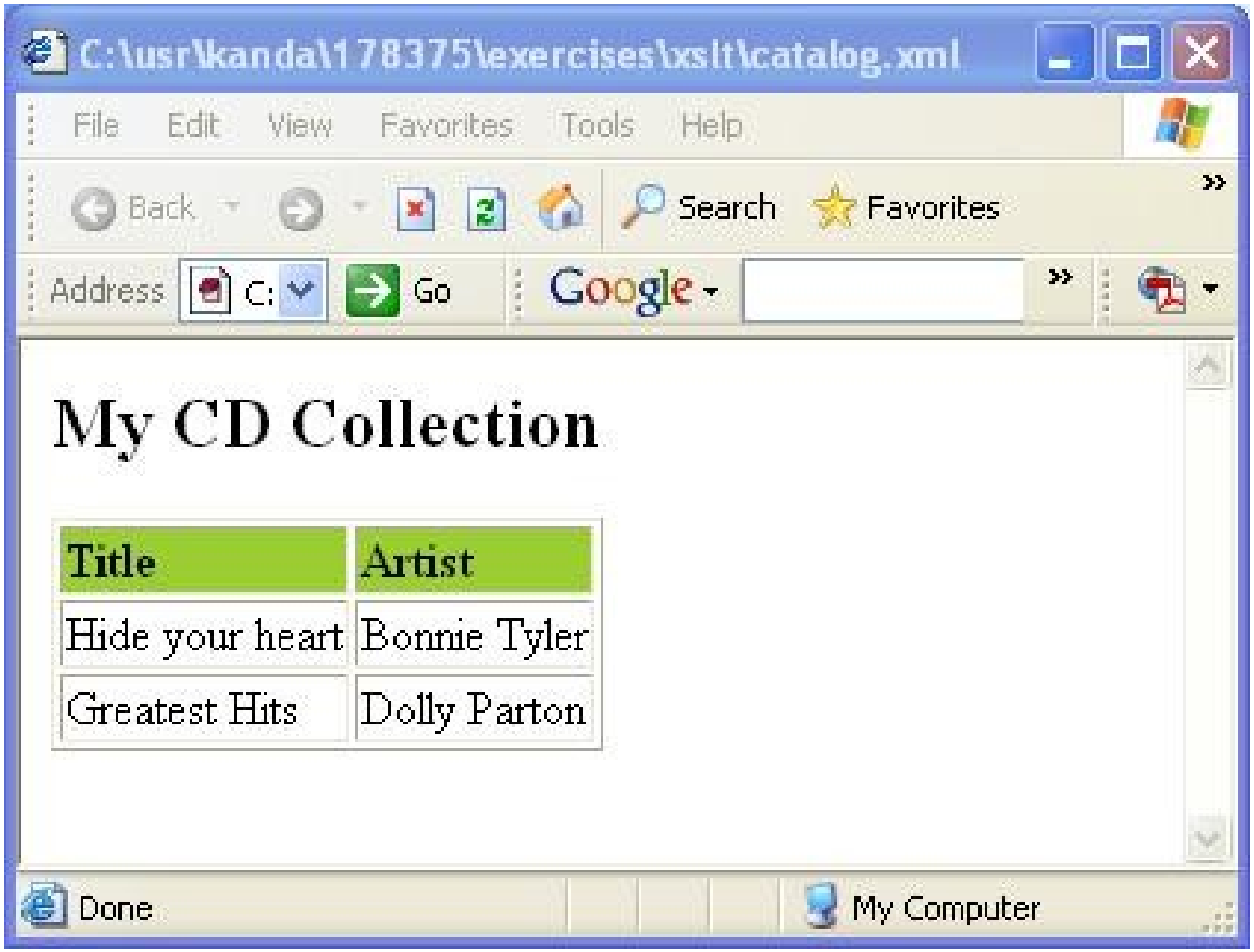
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th align="left">Title</th>
        <th align="left">Artist</th>
      </tr>
```

```
<xsl:for-each select="catalog/cd">
      <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:stylesheet>
```

Example: An Output HTML Document

```
<html>
 <body>
  <h2>My CD Collection</h2>
  <table border="1">
  <tr bgcolor="#9acd32">
  <th align="left">Title</th><th align="left">Artist</th>
  </tr>
  <tr>
  <td>Hide your heart</td><td>Bonnie Tyler</td>
  </tr>
  <tr>
  <td>Greatest Hits</td><td>Dolly Parton</td>
  </tr>
  </table>
 </body>
</html>
```

Example: An Output File on IE View



XSLT Engines (1/2)

- There are several good open source XSLT processors
  - Saxon: The XSLT and XQuery Processor
    - Written by Michael Kay
    - For more info, http://saxon.sourceforge.net/

XSLT Engines (2/2)

- Xalan is an XSLT processor for transforming XML documents into HTML, text, or other XML document types
  - It is developed by Apache XML projects and used in several Apache XML projects including Cocoon
  - For more info, http://xml.apache.org/xalan-j/

Running Xalan Java processor

- Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types
- It can be used from the command line, in an applet or a servlet, or as a module in other programs.
- Sample Usage: assume that xalan.jar is in your CLASSPATH environment
  - java –jar xalan.jar -in catalog.xml –xsl catalog.xsl –out catalog.out

Stylesheets and Templates

- An XSLT processor parses the stylesheet and an input document
- Then it compares the nodes in the input document to the templates in the stylesheet
- When it finds a match, it instantiates the template and adds the result to the output tree

Design an XSLT Stylesheet

- Concentrate on which input constructs map to which output constructs
- Not concentrate on how or when the processor reads the input and generates the output
- XSLT is a push model like SAX rather than a pull model like DOM

Template Rules (1/2)

- An XSLT stylesheet contains examples of what belongs in the output document
- It also contains instructions telling the XSLT processor how to convert input nodes into the example output nodes
- The XSLT processor uses those examples and instructions to convert nodes in the input documents to nodes in the output document

Template Rules (2/2)

- Examples and instructions are written as template rules
- Each template rule has a pattern and a template
- The template rule is represented by an xsl:template element
- The prefix xsl is bound to the namespace URI http://www.w3.org/1999/XSL/Transform

Stylesheets

- A complete XSLT stylesheet is a well-formed XML document
- The root element of the document is xsl:stylesheet which has a version attribute with the value 1.0
- A stylesheet normally contains multiple template rules matching different kinds of input nodes

The First Lines of the XSL File

- <?xml version="1.0"?>
  - Since the style sheet is an XML document itself, the document begins with an XML declaration
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  - The <xsl:stylesheet> tag defines the start of the style sheet
  - Every XSL file needs to specify the XSL namespace so that the XSLT processor knows which version of XSLT to use

Namespaces in XSL

- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  - The namespace prefix xsl: is used in the rest of the XSL file to identify XSL processing statements
  - If a statement is not prefixed with xsl:, then it's simply copied to the output without being processed. This is the way to add text to the output

Minimal but Complete XSLT Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="
   http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Result of an Empty Stylesheet

- Input XML

```
<?xml
  version="1.0"?>
<nation id="th">
  <name>Thailand
  </name>
  <location>
  Southeast Asia
  </location>
</nation>
```

- Output XML

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

Applying an Empty Stylesheet

- The result from applying an empty stylesheet
  - Elements are traversed sequentially
  - Content of each element is put in output
    - Attributes are NOT traversed
  - Default behavior
- Without any specific templates
  - XSLT processor uses the default behavior
- Need to specify templates

XSLT Stylesheet Language

- template
- value-of
- for-each
- sort
- if
- when, choose, otherwise
- apply-templates
- element, attribute
- number
- copy, copy-of
- variable, param
- output

The <xsl:template> Element

<xsl:template match="/">

- Before processing can begin, the part of the XML document with the transformation must be selected with an XPath expression
- The selected section of the document is called a node and is normally selected with the match operator
- If the entire document is to be selected, match the root node using match="/"

The <xsl:value-of> Element

- The <xsl:value-of> element extracts the value of a selected node
- Example (assume we use the “cdcatalog” document)
  - XSL: <td><xsl:value-of select=“catalog/cd/title”/></td>
  - Result: <td>Hide your heart</td>

The <xsl:for-each> Element

- The <xsl:for-each> element allows you to do looping XSL
- It selects every XML element of a specified node set

```
<xsl:for-each select="catalog/cd">
  <tr>
  <td><xsl:value-of select="title"/></td>
  <td><xsl:value-of select="artist"/></td>
  </tr>
</xsl:for-each>
```

Filtering the Output

- We can filter the output from an XML file by adding a criterion to the select attribute in the <xsl:for-each> element

<xsl:for-each
select="catalog/cd[artist='Bonnie Tyler']">

The <xsl:sort> Element

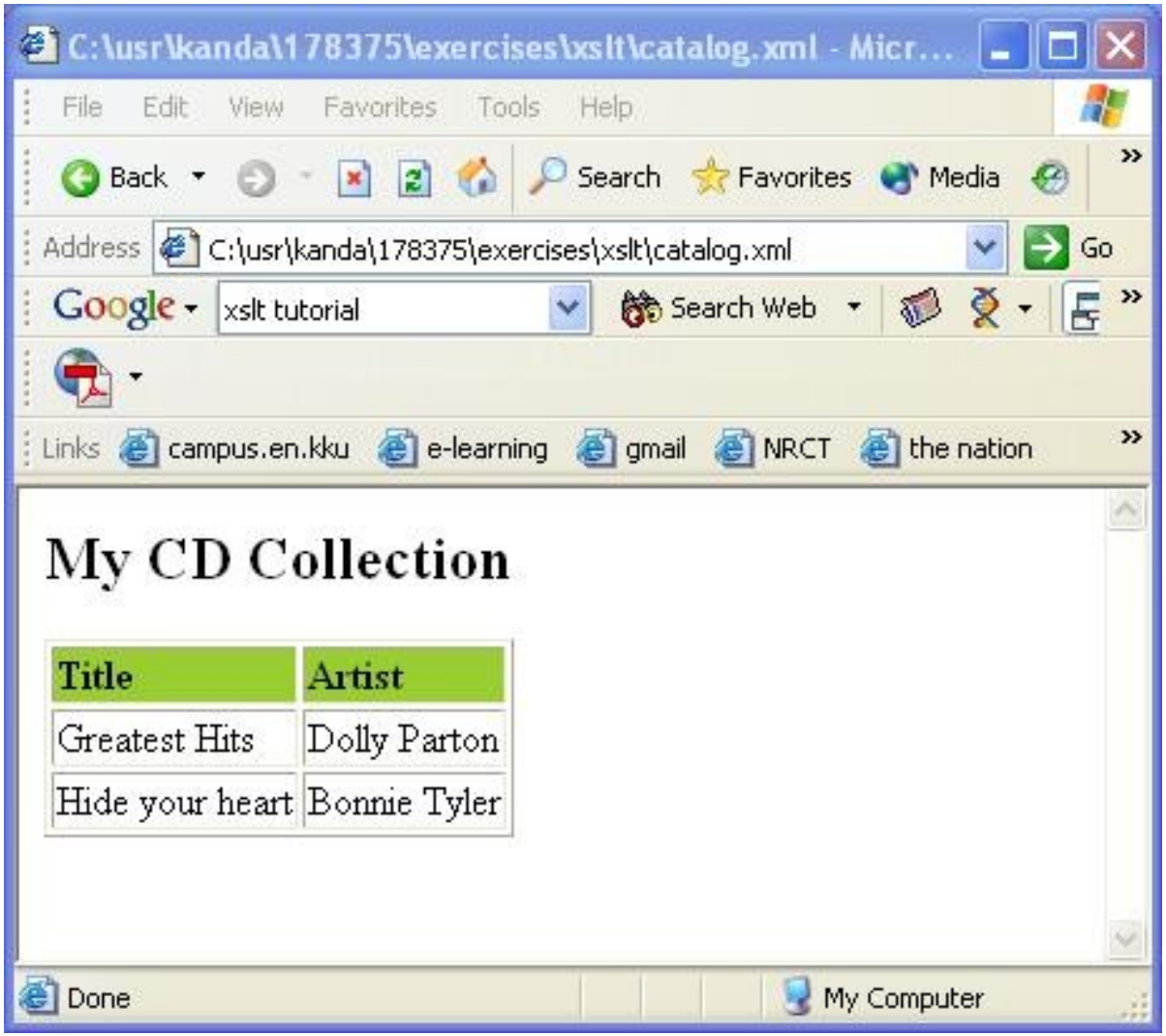
- The <xsl:sort> element is used to sort the output
- The constructs have additional parameters that can be provided
  - order="ascending|descending"
  - data-type="text|number"
  - case-order="upper-first|lower-first"

The <xsl:sort> Element

- The select attribute indicates what XML elements to sort on

```
<xsl:for-each select="catalog/cd">
  <xsl:sort select="title"/>
  <tr>
  <td><xsl:value-of select="title"/></td>
  <td><xsl:value-of
  select="artist"/></td>
  </tr>
</xsl:for-each>
```

After Applying <xsl:sort> on “title”



The <xsl:if> Element

- The <xsl:if> element contains a template that will be applied only if a specified condition is true

  <xsl:if test="price > 10">

  greater than 10

  </xsl:if>

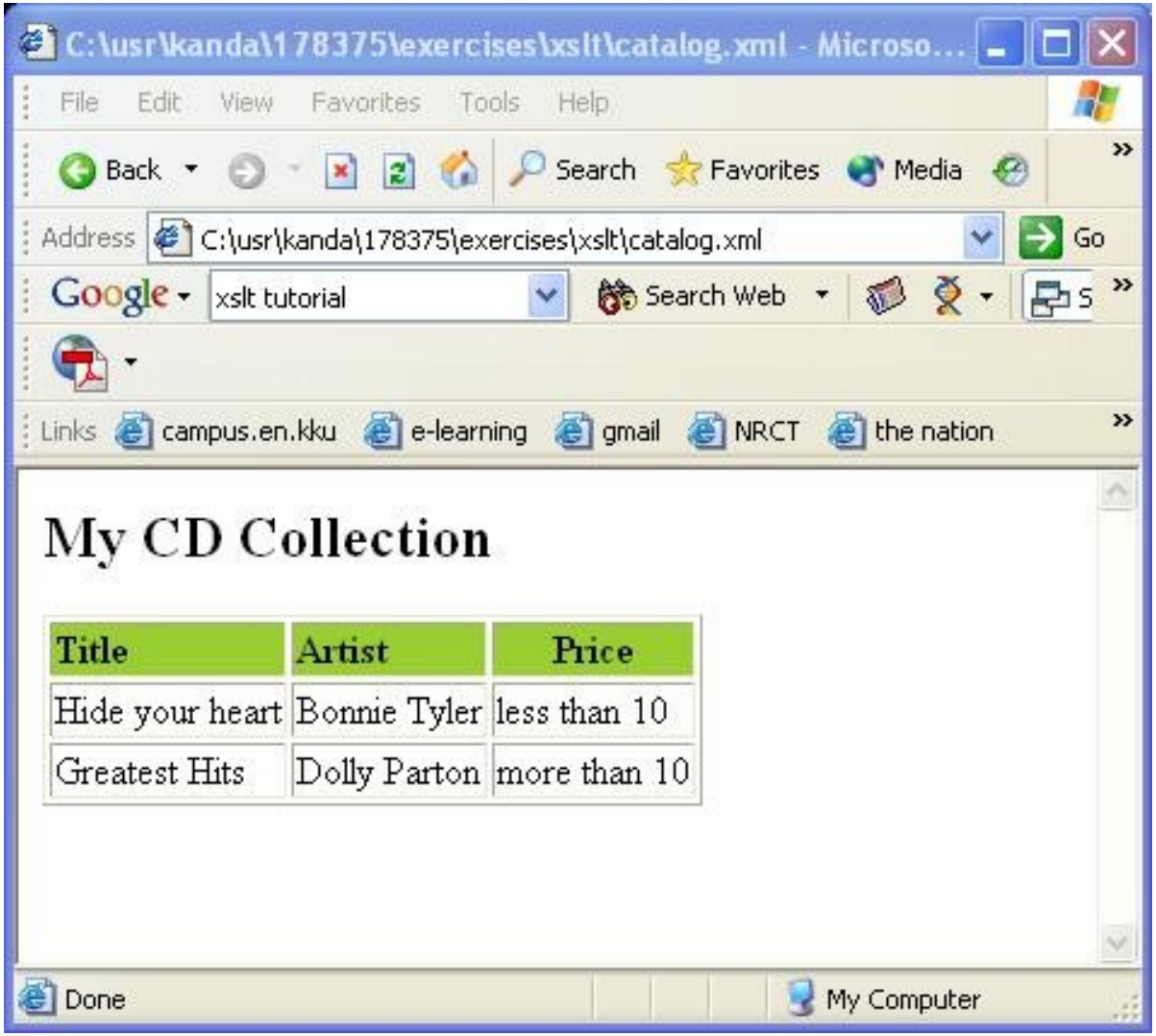
  <xsl:if test="price < 10">

  less than 10

  </xsl:if>

- The value of the requested test attribute contains the expression to be evaluated

After Applying <xsl:if> on “price”



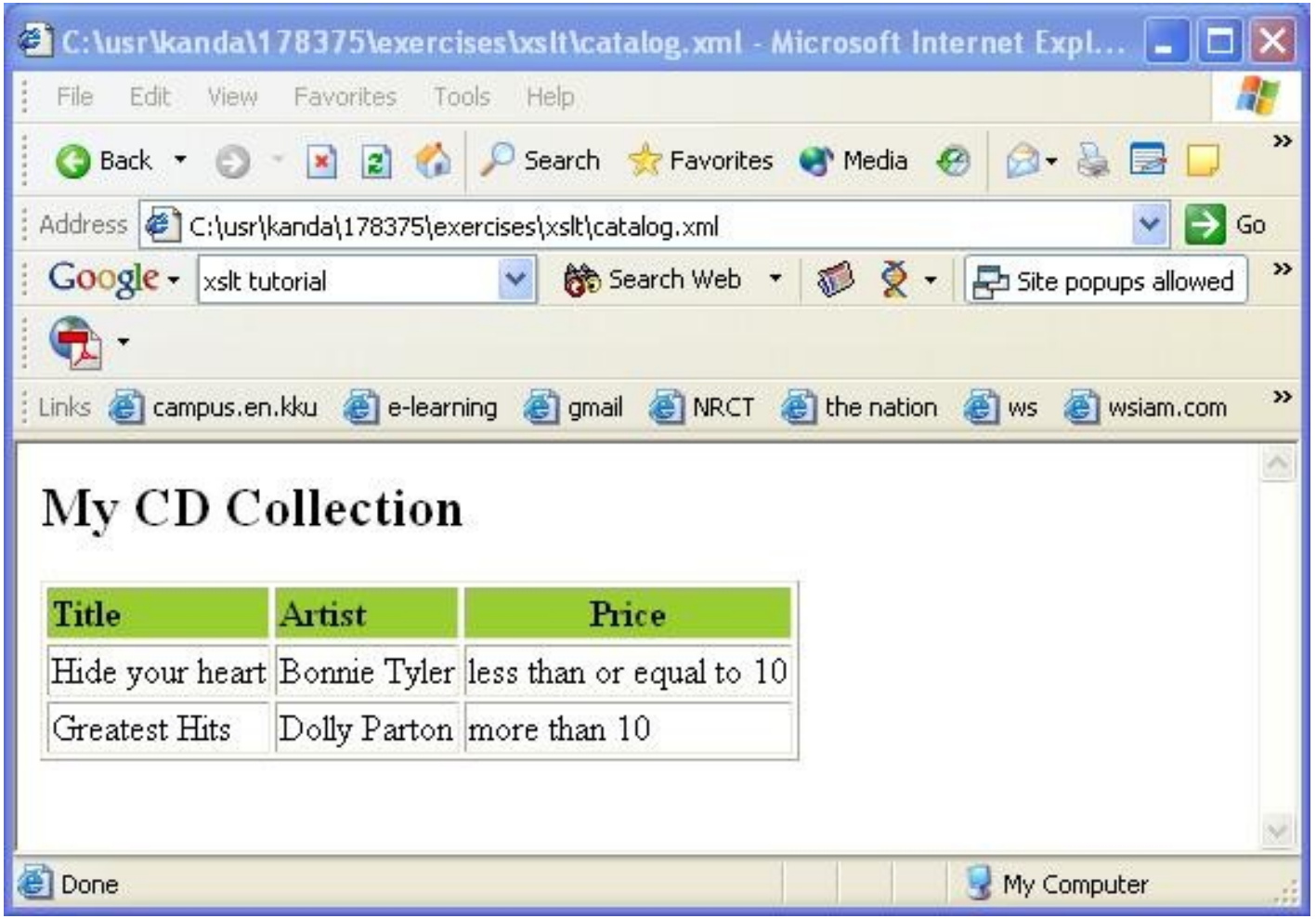
The <xsl:choose> Element

- The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express conditional tests
- If no <xsl:when> element is true, and no <xsl:otherwise> element is present, nothing is created

The <xsl:choose> Element Example

```
<xsl:choose>
  <xsl:when test="price &gt; 10">
   more than 10
  </xsl:when>
  <xsl:otherwise>
   less than or equal to 10
  </xsl:otherwise>
</xsl:choose>
```

After Applying <xsl:choose> on “price”



The <xsl:apply-templates> Element

- XSLT processor reads (traverses) the input XML document sequentially from top to bottom
- Templates are activated in the order they match elements encountered
  - Template for a parent will be activated before the children

The <xsl:apply-templates> Element

- The order of the traversal can be changed by apply-templates
  - It can specify which element or elements should be processed next
  - It can specify an element or elements should be processed in the middle of processing another element
  - It can prevent particular elements from being processed

The <xsl:apply-templates> Element

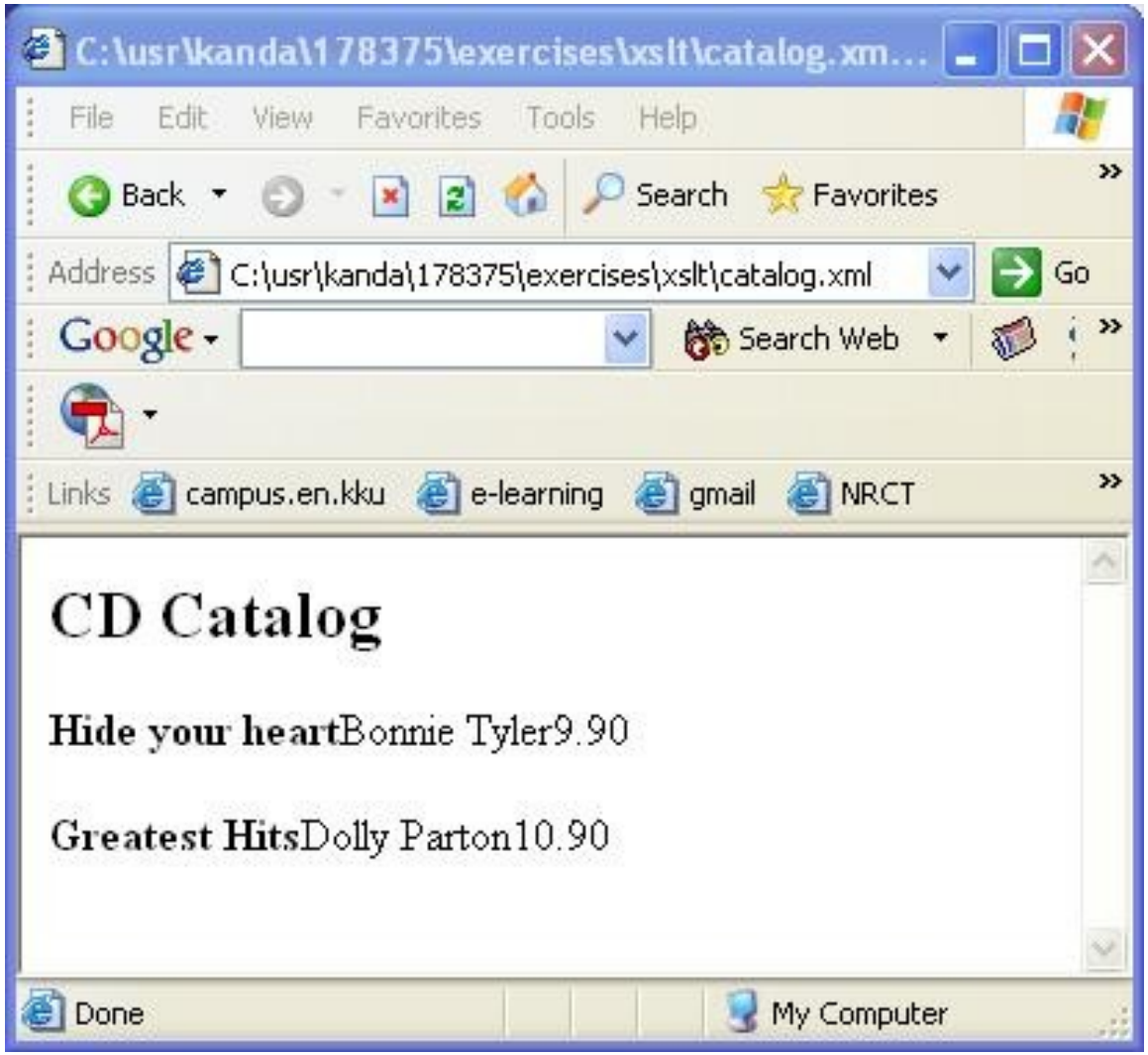
- select attribute contains XPath expression telling the XSLT processor which nodes to process in the input tree
  - The apply-templates with no select attribute means all elements relative to the current element (context node) should be matched

The <xsl:apply-templates/> Element Example

- The <xsl:apply-templates/>: An instruction to apply templates to the children of the current nodes

```
<xsl:template match="cd">
  <p>
  <xsl:apply-templates/>
  </p>
</xsl:templates>
<xsl:template match="title">
  <b><xsl:value-of select="."/></b>
</xsl:template>
```

After Applying <xsl:apply-templates>



Modes

- Same input content needs to appear multiple times in the output document formatted according to different templates
  - Titles of chapters
    - Table of contents
    - In the chapters themselves
- mode attribute
  - xsl:template
  - xsl:apply-templates

nationApplyTemplatesMode.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="nation">
   <xsl:apply-templates select="name" mode="bold"/>
   <xsl:apply-templates select="name"
   mode="italic"/>
</xsl:template>

<xsl:template match="name" mode="bold">
   <b><xsl:value-of select="."/></b>
</xsl:template>
<xsl:template match="name" mode="italic">
   <i><xsl:value-of select="."/></i>
</xsl:template>
</xsl:stylesheet>
```

Applying NationApplyTemplatesMode.xsl

- Input XML

```
<?xml version="1.0"?
  >
<nation id="th">
  <name>Thailand
  </name>
  <location>
  Southeast Asia
  </location>
</nation>
```

- Output

```
 <?xml
  version=“1.0”
encoding=“UTF-
  8”?>
<b>Thailand</b>
  <i>Thailand</i>
```

The <xsl:element> Element

- The <xsl:element> element allows an element to be created with a computed name
- It generates elements in time of processing

<xsl:element name="course">XML and Web Services</xsl:element>

Applying <xsl:element>

- XSL

```
<xsl:template
  match="/">
  <html>
  <xsl:element
  name="h1">The
  Power of
  Love</xsl:element
  >
  </html>
 </xsl:template>
```

- Input

```
<catalog>
  ...
</catalog>
```

- Output

```
<html>
  <h1>The Power of
  Love</h1>
</html>
```

The <xsl:attribute> Element

- The <xsl:attribute> element can be used to add attributes to result elements
- Example

```
<xsl:attribute name="align">
 Center
 </xsl:attribute>
```

Applying <xsl:attribute>

- XSL:

```
<xsl:template
   match="/">
  <html>
  <xsl:element
   name="h1">
   <xsl:attribute
   name="align">center
   </xsl:attribute>
   The Power of
   Love</xsl:element>
  </html>
 </xsl:template>
```

- Input:

```
<catalog>
   ...
</catalog>
```

- Output:

```
<html>
   <h1
   align=“center”>The
   Power of Love</h1>
</html>
```

The <xsl:number>

- The <xsl:number> element is used to insert a formatted number into the result tree
- The number to be inserted may be specified by an expression
- The value attribute contains an expression
- If no value attribute is specified, the xsl:number element inserts a number based on the position of the current node

Applying <xsl:number>

- XSL:

```
<xsl:template match="/">
  <xsl:number
  value=“50000"/>
  <xsl:number
  value="30000"
  grouping-size="3"
  grouping-
  separator=","/>
 </xsl:template>
```

- Input:

```
<catalog>
   ...
</catalog>
```

- Output:

```
50000<br>30,000
```

The <xsl:copy> Element

- The <xsl:copy> element provides an easy way of copying the current node
- The namespace nodes of the current node are automatically copied
- It may have a use-attribute-set attribute to specify and also copy attributes for copied element

Applying <xsl:copy>

XSL:

```
<xsl:template
  match="h1">
  <xsl:copy use-attribute-
  sets="H1">
  <xsl:value-of
  select="."/>
  </xsl:copy>
</xsl:template>
<xsl:attribute-set
  name="H1">
  <xsl:attribute
  name="align">center
  </xsl:attribute>
</xsl:attribute-set>
```

- Input:

```
 <source>
   <h1><i>GREETING
     </i></h1>
   <br/>
</source>
```

- Output:

```
   <h1
     align="center">GR
     EETING</h1>
```

The <xsl:copy-of> Element

- The xsl:copy-of element can be used to insert a result tree fragment into the result tree, without first converting it to string as xsl:value-of does
- But when the result is neither a node-set nor a result tree fragment, the result is converted to a string and then inserted into the result tree

Applying <xsl:copy-of>

XSL:

```
<xsl:template
  match="h1">
    <xsl:copy-of
      select="."/>
<xsl:template>
```

- Input:

```
<source>

    <h1><i>GREETING
    </i></h1>
    <br/>
</source>
```

- Output:

```
<h1><i>GEETING</i
  ></h1>
```

The <xsl:variable> Element

- The <xsl:variable> specify a variable which is a name that may be bound to a value
- The value to which a variable is bound (the value of the variable) can be an object of any of types that can be returned by expressions

Applying <xsl:variable>

- XSL:

```
<xsl:variable
   name="totalChapters">
   <xsl:value-of
   select="count(//chapter)"/>
</xsl:variable>
<xsl:template match="/">
   <xsl:for-each select="//chapter">
        <xsl:value-of select="."/>
        (<xsl:value-of
   select="position()"/>
        /<xsl:value-of
   select="$totalChapters"/>)
   </xsl:for-each>
</xsl:template>
```

- Input:

```
<book>
   <chapter>Chapter
   A</chapter>
   <chapter>Chapter
   B</chapter>
</book>
```

- Output:

Chapter A (1/2)
Chapter B (2/2)

The <xsl:param> Element

- Both <xsl:variable> and <xsl:param> are used to bind variables
- The value specified on the <xsl:variable> cannot be modified later
- However, the value specified on the <xsl:param> variable is only a default value for the binding
- The value specified on the <xsl:param> can be changed later

Applying <xsl:param> (1/2)

- XSL:

```
<xsl:template match="number">
   <xsl:param
   name="type">even
   </xsl:param>
   <xsl:value-of select="."/>
   <xsl:text>(</xsl:text>
   <xsl:value-of select="$type"/>
   <xsl:text>)</xsl:text>
   </xsl:param>
</xsl:template>
```

- Input:

```
<numbers>
  <number>1
  </number>
  <number>4
  </number>
</numbers>
```

- Output:

1 (odd) 4 (even)

Applying <xsl:param> (2/2)

```
<xsl:template match="/">
  <xsl:for-each
  select="//number">
  <xsl:choose>
  <xsl:when
  test="text() mod 2">
  <xsl:apply-templates
  select=".">
  <xsl:with-param
  name="type">odd
  </xsl:with-param>
  </xsl:apply-templates>
  </xsl:when>
```

```
<xsl:otherwise>
  <xsl:apply-
  templates
  select="."/>
</xsl:otherwise>
</xsl:choose>
</xsl:for-each>
</xsl:template>
```

The <xsl:output> Element

- The <xsl:output> element allows stylesheet authors to specify how they wish the result tree to be output
- If an XSLT processor outputs the result tree, it should do so as specified by the <xsl:output> element
- You can specify three types of output documents:
  - XML – This is the default, and such document starts with an <?xml?> declaration
  - HTML – This is standard HTML 4.0
  - Text – This type of output represents pure text

<xsl:output method="html" />

- XSL:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="
  http://www.w3.org/1999/XSL/Transform
  ">
  <xsl:output method="html"/>
  <xsl:template match="/>
  <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

- Input:

```
<?xml version="1.0"?>
<source>
  <h1><i>GREETING</i></h1>
  <br/>
</source>
```

- Output:

```
<source>
  <h1><i>GREETING</i></h1>
  <br>
</source>
```

<xsl:output method="xml"/>

- XSL:

```
<xsl:stylesheet
   version="1.0"
   xmlns:xsl="
   http://www.w3.org/1999/XSL/Transform
   ">
   <xsl:output
   method="xml"/>
   <xsl:template
   match="/>
   <xsl:copy-of
   select="."/>
   </xsl:template>
</xsl:stylesheet>
```

- Input:

```
<?xml version="1.0"?>
<source>
    <h1><i>GREETING</i></h1>
    <br/>
</source>
```

- Output:

```
<?xml version="1.0"
   encoding="UTF-8"?>
<source>
      <h1><i>GREETING</i></h1>
      <br/>
</source>
```

Attributes of <xsl:output>

- Here are some attributes you can use to create or modify XML declarations
  - encoding – Specify the value of XML declarations' encoding attribute
  - doctype-system – Specify an external DTD

<xsl:output encoding=…./>

- XSL:

```
<?xml version="1.0"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.
  w3.org/1999/XSL/Transf
  orm">
  <xsl:output
  encoding="tis-620"/>
  <xsl:template
  match="/">
  <xsl:copy-of
  select="."/>
  </xsl:template>
</xsl:stylesheet>
```

- Input:

```
<?xml version="1.0"?>
<source>
  <h1><i>GREETING</i></h1>
  <br/>
</source>
```

- Output:

```
<?xml version="1.0" encoding="tis-
  620"?>
<source>
  <h1><i>GREETING</i></h1>
  <br/>
</source>
```

<xsl:output doctype-system=…./>

- XSL:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform">
  <xsl:output doctype-system=“source.dtd”/>
  <xsl:template match="/">
  <xsl:copy-of select=“."/>
  </xsl:template>
</xsl:stylesheet>
```

- Input:

```
<?xml version=“1.0”?>
<source>
   <h1><i>GREETING</i>
   </h1>
   <br/>
</source>
```

- Output:

```
<?xml version="1.0"
   encoding="UTF-8"?>
<!DOCTYPE source SYSTEM
   "source.dtd">
<source>
   <h1><i>GREETING</i>
   </h1>
   <br/>
</source>
```

Transformer Programming API (1/2)

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class XSLTSample1 {
    public static void main(String args[]) {
        try {
                TransformerFactory tf =
   TransformerFactory.newInstance();

   /* transformer using instructions in stylesheet
   “nations.xsl” */
          Transformer transformer =
                tf.newTransformer(new
   StreamSource("nations.xsl"));
```

Transformer Programming API (2/2)

```
  /* transformer transforms from input
   “nations.xml” to output “nations.html”*/
          transformer.transform(new
   StreamSource("nations.xml"),
                  new
   StreamResult("nations.html"));
       } catch (Exception ex) {
          ex.printStackTrace(System.err);
       }
    }
}
```

Summary

- XSL is a language for specifying style sheet
- XSLT is an engine for transforming XSL documents and XML documents to an output document
- Every XSL file must have a style sheet element
- Each style sheet element contains several templates
- Different XSL elements have different roles

References

- XSLT, http://www.w3.org/TR/xslt
- XSL-FO http://www.w3.org/TR/xsl
- Sang Shin
  http://www.javapassion.com/xml