TR2

Test Review #2

Note: Test is Comprehensive, Review is Not

Chapter 13 – Exception Handling

- Try ... catch... finally
- Various kinds of exceptions
- The structure of exceptions
  - InnerException
  - Exception Messages
  - etc.

Chapter 14 – GUI Pt1

- Functioning of Various Controls
  - Check Boxes, Radio Buttons, Picture Boxes, etc.
  - Group Boxes and Panels
- Event Driven Programming
  - Creating Event Handlers
- The Use of Message Boxes
  - Icon, title, buttons, etc.
- Anchoring and Docking
- Mouse and Keyboard event handling

Chapter 15 – GUI Pt 2

- Menus
  - Shortcut Keys
- Additional Controls
  - MonthCalendar, DateTimePicker, LinkLabel
  - ListBox, Checked ListBox, ComboBox
  - Tree View Control
- Multi-Document Interface (MDI)
- Timers
  - Animation with timers, etc

Chapter 17 (old version) - Graphics

- Namespace System.Drawing
  - OnPaint, Invalidate static methods
  - Color and Font controls
  - FillRectangle, DrawRectangle, FillElipse, DrawElipse
  - Drawing Arcs
- Using Images
  - Importing image files
  - Painting image files
  - Animating image files, etc.

Chapter 18 – Strings, Characters and Regular Expressions

- Strings and character arrays
- Various string class methods
  - Length, CompareTo, etc, etc
- The String Builder class
- Regular Expressions
  - Applicable classes and there methods
  - Various character classes (user defined also)
  - Various quantifiers
  - Alternation character

Chapter 19 – Using Files

- The use of data streams
  - StreamReader, StreamWriter
  - Files and Directory classes
  - Various File Dialogs
    - OpenFileDialog, SaveFileDialog
- Serialization
  - The various classes needed to perform serialization
  - What it is, how it works (why do it)
  - What can be serialized and what can't

Chapter 20.0 - XML

- What is XML, how is it used, etc.
- How are XML documents defined (validated)
  - DTD – quantifiers, ATTLIST, ELEMENT, types
  - XMLschema – how it works
    - How is it different/better than DTD
  - Take an XML document and give the XML schema and DTD for the document type

20

XML and LINQ to XML

20.3 W3C XML Schema Documents (Cont.)

Automatically Creating Schemas using Visual Studio

- Visual Studio includes a tool that allows you to create a schema from an existing XML document, using the document as a template.
- With an XML document open, select **XML > Create Schema** to use this feature.

Good Programming Practice 20.1

The schema generated by Visual Studio is a good starting point, but you should refine the restrictions and types it specifies so they are appropriate for your XML documents.

20.4 Extensible Stylesheet Language and XSL Transformations

- **Extensible Stylesheet Language** (**XSL**) documents specify how programs are to render XML document data.
- XSL is a group of three technologies—**XSL-FO** (**XSL Formatting Objects**), **XPath** (**XML Path Language**) and **XSLT** (**XSL Transformations**).
  - XSL-FO is a vocabulary for specifying formatting.
  - XPath is a string-based language used by XML to locate structures and data in XML documents.
  - XSL Transformations (XSLT) is a technology for transforming XML documents into other documents.
- XSLT allows you to convert an XML document to an **XHTML** (**Extensible HyperText Markup Language**) document for display in a web browser.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- Transforming an XML document using XSLT involves two tree structures—the **source tree** and the **result tree**.
- XPath is used to locate parts of the source-tree document that match **templates** defined in an **XSL style sheet**.
- When a match occurs, the matching template executes and adds its result to the result tree.
- The XSLT does not analyze every node of the source tree; it selectively navigates the source tree using XSLT's `select` and `match` attributes.
- For XSLT to function, the source tree must be properly structured.

Outline

A Simple XSL Example

- Figure 20.8 lists an XML document that describes various sports.

sports.xml

(1 of 2)

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "sports.xsl"?>

<!-- Fig 20.8: sports.xml -->
<!-- Sports Database -->

<sports>
  <game id = "783">
    <name>Cricket</name>

    <paragraph>
      More popular among Commonwealth nations.
    </paragraph>
  </game>

  <game id = "239">
    <name>Baseball</name>

```

This **processing instruction** (**PI**) specifies the location of the XSL style sheet `sports.xsl`, which will be used to transform the XML document.

Fig. 20.8 | XML document that describes various sports. (Part 1 of 2.)

Outline

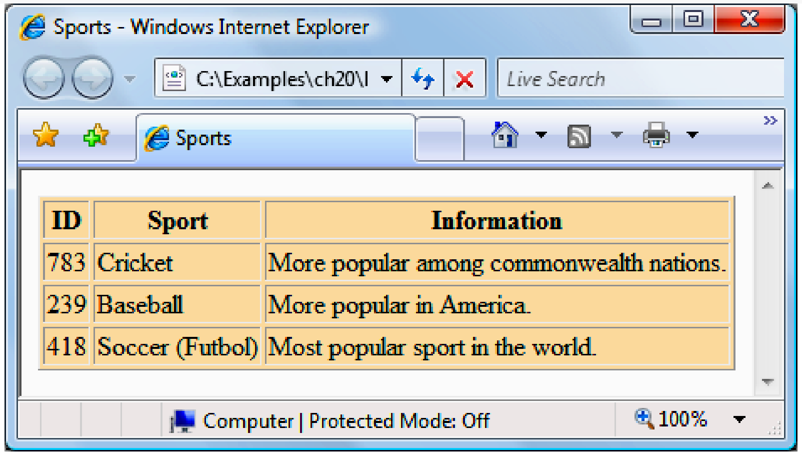
```
         <paragraph>
            More popular in America.
      </paragraph>
   </game>

   <game id = "418">
      <name>Soccer (Futbol)</name>

      <paragraph>
         Most popular sport in the world.
      </paragraph>
   </game>
</sports>
```

sports.xml

(2 of 2)



Sports - Windows Internet Explorer

C:\Examples\ch20\

ID	Sport	Information
783	Cricket	More popular among commonwealth nations.
239	Baseball	More popular in America.
418	Soccer (Futbol)	Most popular sport in the world.

Fig. 20.8 | XML document that describes various sports. (Part 2 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The output shows the result of the transformation (specified in the XSLT template of Fig. 20.9) rendered by Internet Explorer 7.
- To perform transformations, an XSLT processor is required.
- The XML document shown in Fig. 20.8 is transformed into an XHTML document by MSXML when the document is loaded in Internet Explorer.
- MSXML is both an XML parser and an XSLT processor.
- The characters **<?** and **?>** delimit a processing instruction, which consists of a **PI target** and a **PI value**.

Software Engineering Observation 20.4

XSL enables document authors to separate data presentation (specified in XSL documents) from data description (specified in XML documents).

- Figure 20.9 shows the XSL document for transforming the structured data of the XML document of Fig. 20.8 into an XHTML document for presentation.

Outline

sports.xsl

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 20.9: sports.xsl -->
<!-- A simple XSLT transformation -->

<!-- reference XSL style sheet URI -->
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

   <xsl:output method = "xml" omit-xml-declaration = "no"
      doctype-system =
         "http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
      doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>

   <xsl:template match = "/"> <!-- match root element -->

   <html xmlns = "http://www.w3.org/1999/xhtml">
      <head>
         <title>Sports</title>
      </head>

```

The style sheet begins with the `stylesheet` start tag. Attribute `version` specifies the XSLT version.

Attribute `method` is assigned `"xml"`, which indicates that XML is being output to the result tree. Attribute `omit-xml-declaration` specifies that we should not want to omit the XML declaration

Use element `xsl:output` to write an XHTML document type declaration (`DOCTYPE`) to the result tree.

Attributes `doctype-system` and `doctype-public` write the `DOCTYPE` DTD information to the result tree.

Use the `match` attribute to select the **document root** of the XML source document.

The XSLT processor writes this XHTML to the result tree exactly as it appears in the XSL document.

Fig. 20.9 | XSLT that creates elements and attributes in an XHTML document. (Part 1 of 2.)

Outline

```
    <body>
      <table border = "1" style = "background-color: wheat">
        <thead>
          <tr>
            <th>ID</th>
            <th>Sport</th>
            <th>Information</th>
          </tr>
        </thead>

        <!-- insert  each  name and  paragraph  element  value  -->
        <!-- into  a table  row. -->
        <xsl:for-each select = "/sports/game">
          <tr>
            <td><xsl:value-of select = "@id"/></td>
            <td><xsl:value-of select = "name"/></td>
            <td><xsl:value-of select = "paragraph"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>

  </xsl:template>
</xsl:stylesheet>
```

sports.xsl

(2 of 2)

Use element **value-of** to retrieve an attribute's value using the XPath symbol **@**.

Element **xsl:for-each** iterates through the source XML document, searching for the element specified by the **select** attribute.

Fig. 20.9 | XSLT that creates elements and attributes in an XHTML document. (Part 2 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The style sheet begins with the **`stylesheet`** start tag. Attribute **`version`** specifies the XSLT version.
- The `DOCTYPE` identifies XHTML as the type of the resulting document.
- Attributes `doctype-system` and `doctype-public` write the `DOCTYPE` DTD information to the result tree.
- XSLT uses **templates** (i.e., **`xsl:template`** elements) to describe how to transform particular nodes from the source tree to the result tree.
- A template is applied to nodes that are specified in the `match` attribute.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The XPath character `/` (a forward slash) is used as a separator between element names.
- In XPath, a leading forward slash specifies that we are using **absolute addressing** (i.e., we are starting from the root).
- Element `xsl:for-each` iterates through the source XML document, searching for the element specified by the `select` attribute.
- Use element `value-of` to retrieve an attribute's value using the XPath symbol `@`.
- When an XPath expression has no beginning forward slash, the expression uses **relative addressing**.

Outline

Using XSLT to Sort and Format Data

- Figure 20.10 presents an XML document (`sorting.xml`) that marks up information about a book.

sorting.xml

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig20.10: sorting.xml -->
<!-- XML document containing book information -->

<?xml-stylesheet type = "text/xsl" href = "sorting.xsl"?>

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

   <author>
      <firstName>Jane</firstName>
      <lastName>Blue</lastName>
   </author>

   <chapters>
```

Fig. 20.10 | XML document containing book information. (Part 1 of 2.)

Outline

sorting.xml

(2 of 2)

```
         <frontMatter>
            <preface  pages = "2" />
         <contents pages = "5" />
         <illustrations pages = "4" />
      </frontMatter>

      <chapter number = "3" pages = "44">Advanced XML</chapter>
      <chapter number = "2" pages = "35">Intermediate XML</chapter>
      <appendix number = "B" pages = "26">Parsers and Tools</appendix>
      <appendix number = "A" pages = "7">Entities</appendix>
      <chapter number = "1" pages = "28">XML Fundamentals</chapter>
   </chapters>

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

Fig. 20.10 | XML document containing book information. (Part 2 of 2.)

- XSL style sheet can sort an XML file’s data for presentation purposes.

Outline

- Figure 20.11 presents an XSL document (`sorting.xsl`) for transforming `sorting.xml` (Fig. 20.10) to XHTML.

`sorting.xsl`

(1 of 5)

```
<?xml version = "1.0"?>
<!-- Fig. 20.11: sorting.xsl -->
<!-- Transformation of book information into XHTML -->

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

   <!-- write XML declaration and DOCTYPE DTD information -->
  <xsl:output method = "xml" omit-xml-declaration = "no"
     doctype-system = "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
     doctype-public = "-//W3C//DTD XHTML 1.1//EN"/>

  <!-- match document root -->
  <xsl:template match = "/">
     <html>
        <xsl:apply-templates/>
     </html>
  </xsl:template>

```

The `<xsl:apply-templates/>` element specifies that the XSLT processor is to apply the `xsl:template`s defined in this XSL document to the current node's children.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 1 of 5.)

Outline

```
<!-- match book -->
<xsl:template match = "book">
  <head>
     <title>ISBN <xsl:value-of select = "@isbn"/> -
       <xsl:value-of select = "title"/></title>
  </head>

  <body>
     <h1 style = "color: blue"><xsl:value-of select = "title"/></h1>
     <h2 style = "color: blue">by
        <xsl:value-of select = "author/firstName"/>
        <xsl:text> </xsl:text>
       <xsl:value-of select = "author/lastName"/>
     </h2>

     <table style = "border-style: groove; background-color: wheat">

       <xsl:for-each select = "chapters/frontMatter/*">
          <tr>
             <td style = "text-align: right">
                <xsl:value-of select = "name()"/>
             </td>
```

sorting.xsl

(2 of 5)

Use the `book`'s ISBN (from attribute `isbn`) and the contents of element `title` to create the string that appears in the browser window's title bar.

The `xsl:text` element is used to insert literal text.

Select each element (indicated by an asterisk) that is a child of element `frontMatter`.

Node-set function `name` retrieves the current node's element name.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 2 of 5.)

Outline

```

                           <td>
                     <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>

            <xsl:for-each select = "chapters/chapter">
              <xsl:sort select = "@number" data-type = "number"
                 order = "ascending"/>
               <tr>
                  <td style = "text-align: right">
                     Chapter <xsl:value-of select = "@number"/>
                  </td>

                  <td>
                     <xsl:value-of select = "text()"/>
                     ( <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>
```

sorting.xsl

(3 of 5)

Use element `xsl:sort` to sort the selected elements by the value given by its `select` attribute.

Use **node-set function** `text` to obtain the text between the `chapter` start and end tags.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 3 of 5.)

Outline

```
      <xsl:for-each select = "chapters/appendix">
         <xsl:sort select = "@number" data-type = "text"
            order = "ascending"/>
         <tr>
            <td style = "text-align: right">
               Appendix <xsl:value-of select = "@number"/>
            </td>

            <td>
               <xsl:value-of select = "text()"/>
               ( <xsl:value-of select = "@pages"/> pages )
            </td>
         </tr>
      </xsl:for-each>
   </table>

```

sorting.xsl

(4 of 5)

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 4 of 5.)

Outline

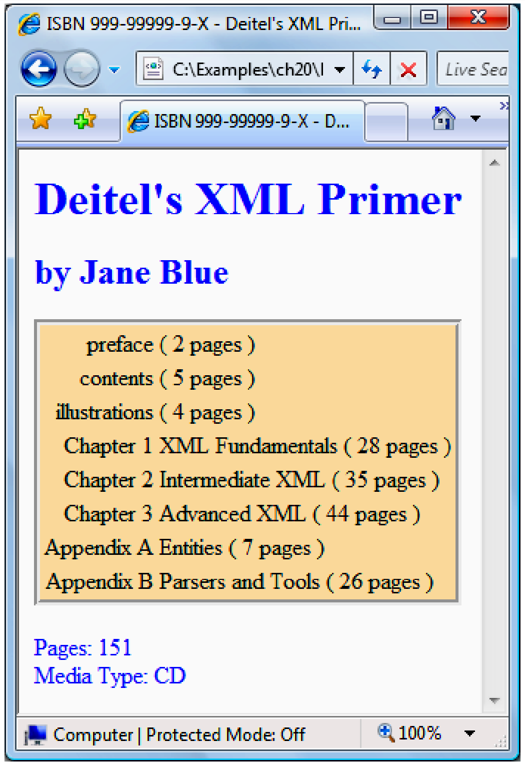
```
         <p style = "color: blue">Pages:
            <xsl:variable name = "pagecount"
               select = "sum(chapters//*/@pages)"/>
            <xsl:value-of select = "$pagecount"/>
         <br />Media Type: <xsl:value-of select = "media/@type"/></p>
      </body>
   </xsl:template>
</xsl:stylesheet>
```

sorting.xsl

(5 of 5)

Use an **XSL variable** to store the value of the book's total page count and output the page count to the result tree.

Function **`sum`** totals a set of values found using XPath.


ISBN 999-99999-9-X - Deitel's XML Pri...
C:\Examples\ch20\I
Live Sea
ISBN 999-99999-9-X - D...
Computer | Protected Mode: Off
100%

Deitel's XML Primer

by Jane Blue

preface (2 pages)
contents (5 pages)
illustrations (4 pages)
Chapter 1 XML Fundamentals (28 pages)
Chapter 2 Intermediate XML (35 pages)
Chapter 3 Advanced XML (44 pages)
Appendix A Entities (7 pages)
Appendix B Parsers and Tools (26 pages)

Pages: 151
Media Type: CD

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 5 of 5.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The `<xsl:apply-templates/>` element specifies that the XSLT processor is to apply the `xsl:template`s defined in this XSL document to the current node's children.
- The **`xsl:text`** element is used to insert literal text.
- **Node-set function `name`** retrieves the current node's element name.
- Use element **`xsl:sort`** to sort the selected elements by the value given by its **`select`** attribute.
- Attribute **`data-type`**, with value `"number"`, specifies a numeric sort. This attribute also accepts the value `"text"`.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- Attribute `order` can be set to either `"ascending"` or `"descending"`.
- XSL variables cannot be modified after they are initialized.
  - Attribute `name` specifies the variable's name
  - Attribute `select` assigns a value to the variable.
- Function `sum` totals a set of values found using XPath.
- Two slashes in an XPath expression indicate **recursive descent**.

Performance Tip 20.1

Selecting all nodes in a document when it is not necessary slows XSLT processing.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

Summary of XSL Style-Sheet Elements

- Figure 20.12 lists commonly used XSL elements.

Element	Description
`<xsl:apply-templates>`	Applies the templates of the XSL document to the children of the current node.
`<xsl:apply-templates match ="`*expression*`">`	Applies the templates of the XSL document to the children of the nodes matching *expression*. The value of the attribute match (i.e., *expression*) must be an XPath expression that specifies elements.
`<xsl:template>`	Contains rules to apply when a specified node is matched.
`<xsl:value-of select ="`*expression*`">`	Selects the value of an XML element and adds it to the output tree of the transformation. The required `select` attribute contains an XPath expression.

Fig. 20.12 | XSL style-sheet elements. (Part 1 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

Element	Description
`<xsl:for- each select = "`*expression*`">`	Applies a template to every node selected by the XPath specified by the `select`attribute.
`<xsl:sort select = "`*expression*`">`	Used as a child element of an `<xsl:apply-templates->` or `<xsl:for-each>` element. Sorts the nodes selected by the `<xsl:apply-template>` or `<xsl:for-each>` element so that the nodes are processed in sorted order.
`<xsl:output>`	Has various attributes to define the format (e.g., XML, XHTML), version (e.g., 1.0, 2.0), document type and MIME type of the output document. MIME types are discussed in Section 22.2. This tag is a top-level element—it can be used only as a child element of an `xsl:stylesheet`.
`<xsl:copy>`	Adds the current node to the output tree.

Fig. 20.12 | XSL style-sheet elements. (Part 2 of 2.)