



Урок 5

План заняття:

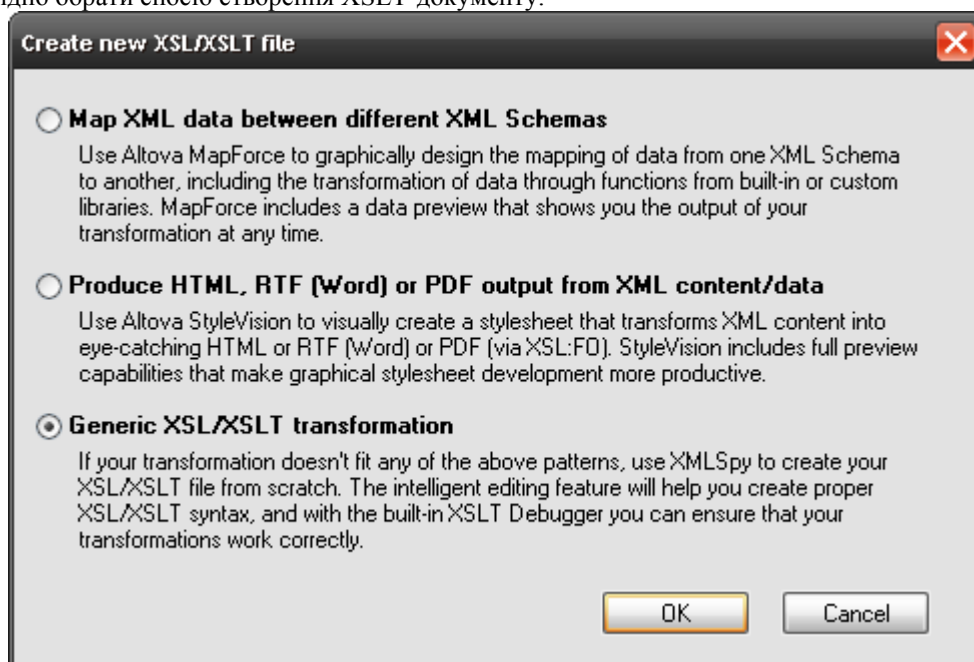
1. Трансформація XML-документів з використанням мови XSLT. Створення XSLT документа. Шаблонні правила
2. Практичний приклад XSLT трансформації xml-html. Підтримка XSLT в Web-браузерах
3. Елементи XSLT
 - 3.1. Рекурсивний виклик шаблонів. Елемент `xsl:apply-templates`
 - 3.2. Елементи `xsl:attribute` та `xsl:element`
 - 3.3. Умови та цикли. Елементи `xsl:if`, `xsl:for-each` та `xsl:choose`
 - 3.4. Сортування. Елемент `xsl:sort`
 - 3.5. Копіювання вузлів. Елементи `xsl:copy` та `xsl:copy-of`
 - 3.6. Коментарі та текстові вузли
 - 3.7. Створення інструкцій по обробці
4. Змінні в XSLT
5. Іменовані шаблони та параметри
6. Домашнє завдання

1. Трансформація XML-документів з використанням мови XSLT. Створення XSLT документа. Шаблонні правила

На минулій парі ми почали розгляд мови трансформацій XML документів - XSLT, але зупинились при її розгляді на мові XPath, без якої трансформація неможлива. Призначенням мови виразів XPath являється звернення до частин XML-документів з метою здійснення вибірки необхідних вузлів. Ці вибірки можна використовувати для подальшої конвертації обраних вузлів за допомогою мови XSLT. Оскільки мова XPath вивчена, то прийшов час повернутись та розглянути в повній мірі XSLT.

Нагадаємо, що мова трансформацій XSLT (XML Stylesheet for Transformation) – це мова конвертування структури документів. XSLT являється реалізацією XML, тобто це XML-мова в повному сенсі цього слова, оскільки програми на XSLT являються добре оформленими XML-документами, які містять правила трансформації. Але, по традиції, документ, написаний на мові XSLT, називається **таблицею стилів (stylesheet)**, а сам файл має розширення **.xsl** або **.xslt**.

Отже, розпочнемо. Для початку необхідно створити XSLT-документ. Як і у випадку створення XML-документів та схем, Ви можете для цього скористатись будь-яким текстовим редактором, але, якщо Ви обрали для цієї цілі редактор Altova XML Spy, то в такому разі після вибору пункту меню **File->New->XSLT Stylesheet**, перед вами з'явиться діалогове вікно, в якому необхідно обрати спосіб створення XSLT-документу.



Перших два пункта передбачають створення вже готового XSLT-документу і тому вони нам не підходять. Останній же пункт передбачає створення пустого документа для трансформації, і оскільки ми хочемо чогось навчитись, то його і оберемо.



Як і кожен XML-документ, XSLT-документ повинен починатись з XML-декларації:

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

Після цього необхідно вказати кореневий елемент, яким для документа XSLT (таблиці стилів) є елемент **xsl:stylesheet** або **xsl:transform**. Ці елементи повністю ідентичні, однак останній вважається застарілим і зараз практично не використовується.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  [id="ідентифікатор"]
  [extension-element-prefixes="префікси"]
  [exclude-result-prefixes="префікси"]>
</xsl:stylesheet>
```

Кореневий елемент має один обов'язковий атрибут **version**, що вказує на версію мови XSLT, в відповідності до якої був побудований документ. Останньою версією XSLT являється XSLT 2.0, яка вийшла 23 січня 2007 року. Але, не дивлячись на це більшість розробників вказують версію 1.0, оскільки процесори XSLT 2.0 в такому випадку розуміють, що необхідно забезпечити зворотню сумісність з версіями XSLT 1.0.

Крім атрибута **version** потрібно визначити простір імен для елементів мови XSLT з іменем **xsl** (як правило), якому відповідає URL-адреса <http://www.w3.org/1999/XSL/Transform>. Значення 1999 в URI просторі імен XSLT вказує на рік затвердження специфікації консорціумом W3. Значення XSL вказує на те, що це специфікація мови XSL, а Transform визначає приналежність до трансформації.

Необов'язковий атрибут **id** може містити унікальний ідентифікатор даної таблиці стилів. Цей атрибут здебільшого використовується лише, коли одна таблиця стилів включається в іншу.

Необов'язковий атрибут **extension-element-prefixes** дозволяє вказати список префіксів просторів імен, які будуть використовуватись для елементів розширення XSLT документа. Розширення XSLT та XPath використовується для включення за допомогою функцій або елементів програмного коду (класів, процедур або функцій), написаного на інших мовах програмування (Java, JavaScript, Python тощо).

Необов'язковий атрибут **exclude-result-prefixes** містить список просторів імен, визначення яких не потрібно включати в вихідний документ.

Отже, мінімальний і класичний каркас таблиці стилів XSLT буде мати наступний вигляд:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Після цього потрібно вказати послідовність дій, які необхідно виконати для досягнення результату. Конвертування являє собою набір шаблонних правил, кожне з яких визначає процедуру обробки певної частини XML-документа. Іншими словами, таблиці стилів в XSLT оголошують правила трансформації, які XSLT процесор застосовує до вхідного документа і в результаті яких він генерує вихідний документ.

Ці правила оголошуються в середині кореневого елемента і для цього в мові XSLT існує близько 3,5 десятків елементів. Елемент **xsl:stylesheet** серед всіх елементів, може включати в себе наступні:

- ✓ **xsl:import;**
- ✓ **xsl:include;**
- ✓ **xsl:strip-space;**
- ✓ **xsl:output;**
- ✓ **xsl:key;**
- ✓ **xsl:decimal-format;**
- ✓ **xsl:namespace-alias;**
- ✓ **xsl:attribute-set;**
- ✓ **xsl:variable;**
- ✓ **xsl:param;**
- ✓ **xsl:template.**

Отже, всі ці елементи можуть бути елементами верхнього рівня. Більше того, їх оголошення, крім **xsl:variable** та **xsl:param** повинне знаходитись лише на верхньому рівні, тобто після кореневого елемента та перед визначенням шаблонів трансформації. При цьому порядок слідування немає значення, але у випадку імпортування зовнішніх документів, першим дочірнім елементом обов'язково повинен бути **xsl:import**.

Серед цих елементів варто було б відмітити елемент **xsl:output**, який вказує на те, як буде здійснюватись вивід дерева результуючого документа.

```
<xsl:output method = "xml | html | text | префікс:ім'я"
  [version = "версія"]
  [encoding = "кодова сторінка"]>
```



```
[omit-xml-declaration = "yes | no"]
[standalone = "yes | no"]
[doctype-public = "публічний ідентифікатор"]
[doctype-system = "системний ідентифікатор"]
[cdata-section-elements = "імена"]
[indent = "yes | no"]
[media-type = "медіа-тип"] />
```

Як видно з синтаксису, даний елемент має один обов'язковий атрибут **method**, який задає спосіб трансформації. Значенням даного атрибута може бути довільне ім'я, але технічна рекомендація визначає лише три значення: **text**, **html** або **xml**. Якщо в таблиці стилів відсутній елемент `xsl:output`, то значення по замовчужанню визначається, виходячи з наступної умови: якщо корінь вхідного документу має дочірній елемент з іменем «html», перед яким вказуються лише символи пропуску, то методом трансформації приймається «html». У всіх інших випадках методом трансформації по замовчужанню являється «xml».

Варто також зауважити, що якщо результатом документом буде XHTML сторінка, то ефективніше задавати метод виводу «xml», а не «html».

Необов'язкові атрибути:

- **version** - задає версію вихідного документа;
- **encoding** - вказує на кодову сторінку, яка буде використана при виведенні даних;
- **indent** - задає можливість встановлення відступів між тегами в результаті виведення документа для більшої наочності;
- **media-type** – тип вмісту (MIME-тип) вихідного документа;
- **omit-xml-declaration** (пропустити декларацію XML) – вказує на необхідність включати XML-декларацію в вихідний документ;
- **standalone** (самостійний документ) – вказує на необхідність виводити оголошення на самостійність, автономність документа;
- **doctype-public** – вказує ідентифікатор PUBLIC для використовуваного документа DTD;
- **doctype-system** - вказує системний ідентифікатор SYSTEM для використовуваного документа DTD;
- **cdata-section-elements** – список елементів, текстовий вміст яких повинні бути виведені в розділі CDATA.

Використання вищеописаних атрибутів залежить від прийнятого способу трансформації. Крім того, таблиця стилів може містити кілька елементів `xsl:output` і включати в себе інші таблиці стилів. В такому випадку, всі елементи `xsl:output`, які зустрічаються в таблиці стилів, зливаються в єдиний діючий елемент `xsl:output`. Якщо у атрибута кілька таких значень, то буде згенерована помилка.

Отже, при додаванні способу трансформації до нашого XSLT документу, він набуде наступного вигляду:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>
</xsl:stylesheet>
```

Всі підготовчі процеси проведено, тепер час писати власні шаблони для трансформації. Всі шаблони розміщуються під елементом **xsl:output** за допомогою елемента **xsl:template**, тобто він дозволяє задати **шаблонне правило (template rule)** для трансформації.

```
<xsl:template {match="шаблон XPath" | name="ім'я"}
  [priority = "пріоритет"]
  [as = "тип результату"]
  [mode = "режим"]>
  Конструктор послідовності
</xsl:template>
```

Атрибут **match** задає шаблон, зразок (pattern) для відбору вузлів, які підлягають трансформації. Атрибут **name** дозволяє вказати ім'я шаблону, перетворюючи його в **іменованний шаблон (named template)**. Ім'я шаблону – це звичайне розширене ім'я XML типу QName. Після того, як шаблон має ім'я, його можна викликати по імені.

Зауважимо, що кожен з атрибутів **match** та **name** не є обов'язковими, але один з них повинен бути вказаний.

В елементі `xsl:template` існує також **3 необов'язкові атрибути**:

- **priority** – назначає шаблону пріоритет, який буде враховуватись при відборі правил, що застосовуються до даного вузла. По суті, пріоритет – це просто число (додатне або від'ємне); при цьому, чим більше число, тим вищий пріоритет.
- **mode** – визначає режим обробки даного шаблону. Режими дозволяють задавати різні приведення для одних і тих же частин документа. Типовим прикладом такої необхідності являється генерація заголовка документа разом з трансформацією його вмісту. Існує кілька особливостей його використання:
 - Якщо елемент `xsl:template` не має атрибута `match`, то у нього не може бути атрибута `mode`;



- Якщо елемент `xsl:apply-templates` має атрибут `mode`, то він відноситься лише до тих шаблонних правил `xsl:template`, які мають атрибути `mode` з тим же значенням.
- **as** – вказує на бажаний тип результату. При цьому отримана послідовність буде приведена до даного типу.

В самому тілі шаблону міститься **конструктор послідовності (sequence constructor)** вузлів та значень, який і буде результатом трансформації відібраних по шаблону даних.

Наприклад, існує наступний XML-документ, що містить інформацію про деяку компанію.

```
<company name="Logo Inc.">
</company>
```

Наприклад, необхідно написати шаблон, який виводить у вихідний HTML-документ назву компанії, яка розміщується в атрибуті `name`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="company">
    Назва компанії: <xsl:value-of select="./@name" />
  </xsl:template>
</xsl:stylesheet>
```

Щодо згаданого вище елемента **xsl:value-of**, то цей елемент являється найчастіше використовуваним елементом XSLT. Він служить для обрахунку значень виразу, записаного в його обов'язковому атрибуті `select`, і трансформує його в рядок. Повний синтаксис даного елемента наступний:

```
<xsl:value-of
  select = "вираз"
  [disable-output-escaping = "yes | no"]>
</xsl:value-of>
```

Результатом роботи даного елемента буде текстовий вузол, який виведе на екран всі спеціальні символи (якщо вони були отримані в результаті обрахунку). Якщо це не потрібно, тобто ви бажаєте, щоб спеціальні символи були замінені відповідними символічними або вбудованими сутностями, то в необов'язковому атрибуті **disable-output-escaping** варто вказати значення **yes**.

Наприклад, результатом виконання елемента

```
<xsl:value-of select="concat('hello ', '&', ' buy')"/>
```

буде текстовий вузол

```
hello & buy
```

А результатом виконання наступного шаблону

```
<xsl:value-of select="concat('hello ', '&', ' buy')"
  disable-output-escaping = "yes" />
```

буде текст

```
hello & buy
```

Оскільки елемент `disable-output-escaping="yes"` може призводити до створення документів в невірному форматі, то його слід застосовувати лише у випадку крайньої необхідності. Адже, невірний формат виводу при певних обставинах може призвести до помилки.

Наведемо ще один приклад застосування елемента `xsl:value-of`. Припустимо, що елемент `company` містить дочірній елемент `year`, що вказує на рік заснування компанії.

```
<company name="Logo Inc.">
  <year>2000</year>
</company>
```

А тепер розширимо наш шаблон і під назвою компанії виведемо рік її заснування:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>
```



```
<xsl:template match="company">
  Назва компанії: <xsl:value-of select="./@name" /><br/>
  Рік заснування: <xsl:value-of select="/year" />
</xsl:template>

</xsl:stylesheet>
```

На місці елемента `xsl:value-of` підставиться текст **2000**, тобто буде зроблена текстова підстановка.

2. Практичний приклад XSLT трансформації xml-html. Підтримка XSLT в Web-браузерах

А тепер перейдемо безпосередньо до практики, щоб краще зрозуміти всі тонкощі трансформації за допомогою мови XSLT. Як ви вже знаєте, існує три види трансформації вхідного документа, але кожен з них має свої нюанси і тому варто розглянути кожен окремо. Розпочнемо наше знайомство з самого простого виду трансформації – це xml-html.

Отже, для початку опишемо вхідний XML документ. Нехай він містить інформацію про студентів певної групи та матиме наступний вигляд:

students.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<group name="7HC15-5H" spec="Programming">
  <student name="Vasya Pupkin"> I am Vasya. I am the best programmer</student>
  <student name="Olya Pupkina"> I am Olga. I am the best designer</student>
  <student name="Innokentiy Pupkin"> I am Innokentiy. Simply Innokentiy</student>
</group>
```

Після цього спробуємо написати XSLT-документ, який буде містити правила трансформації даного документа в HTML-документ, який буде виводити в табличному вигляді дані про студентів, а перед таблицею виведе назву групи та спеціалізацію.

students.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/> <!-- вказуємо метод трансформації -->
  <xsl:template match="/"> <!-- шаблон для кореневого елемента -->
    <!-- Основний html документ. Його тіло -->
    <html>
      <head>
        <title>groups demo</title>
      </head>
      <body>
        <xsl:apply-templates /> <!-- застосувати на цьому місці інші
                                   (описані нижче) шаблони (рекурсивно) -->
      </body>
    </html>
  </xsl:template>

  <!-- Трансформація документа. Опис вмісту -->
  <xsl:template match="group"> <!-- Обробляємо тег group - кореневий елемент -->
    <h1>GROUP: <xsl:value-of select="./@name" /></h1> <!-- назва групи -->
    <font color="#FF0000">
      Specialization: <xsl:value-of select="./@spec"/> <!-- спеціалізація -->
    </font>

    <!-- Будуємо таблицю з інформацією про студентів -->
    <table border="1" style="bordercolor:#FF0000;">
      <tbody>
        <tr>
          <th>NameStudent</th><th>Description</th>
        </tr>
        <xsl:apply-templates /><!-- застосувати на цьому місці інші
                                   (описані нижче) шаблони (рекурсивно) -->
      </tbody>
    </table>
  </xsl:template>
```



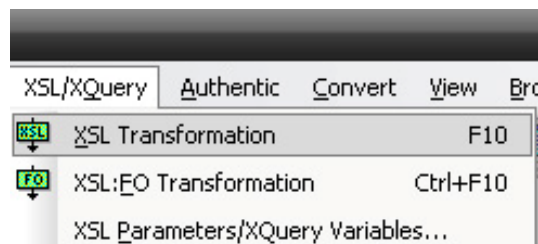
```

<!-- Створюємо шаблон для трансформації даних про кожного студента -->
<xsl:template match="student"> <!-- Обробляємо теги student -->
    <tr>
        <td><xsl:value-of select="@name"/></td> <!-- ім'я та прізвище -->
        <td><xsl:value-of select="."/></td> <!-- Текстове значення
                                           контекстного вузла -->
    </tr>
</xsl:template>
</xsl:stylesheet>

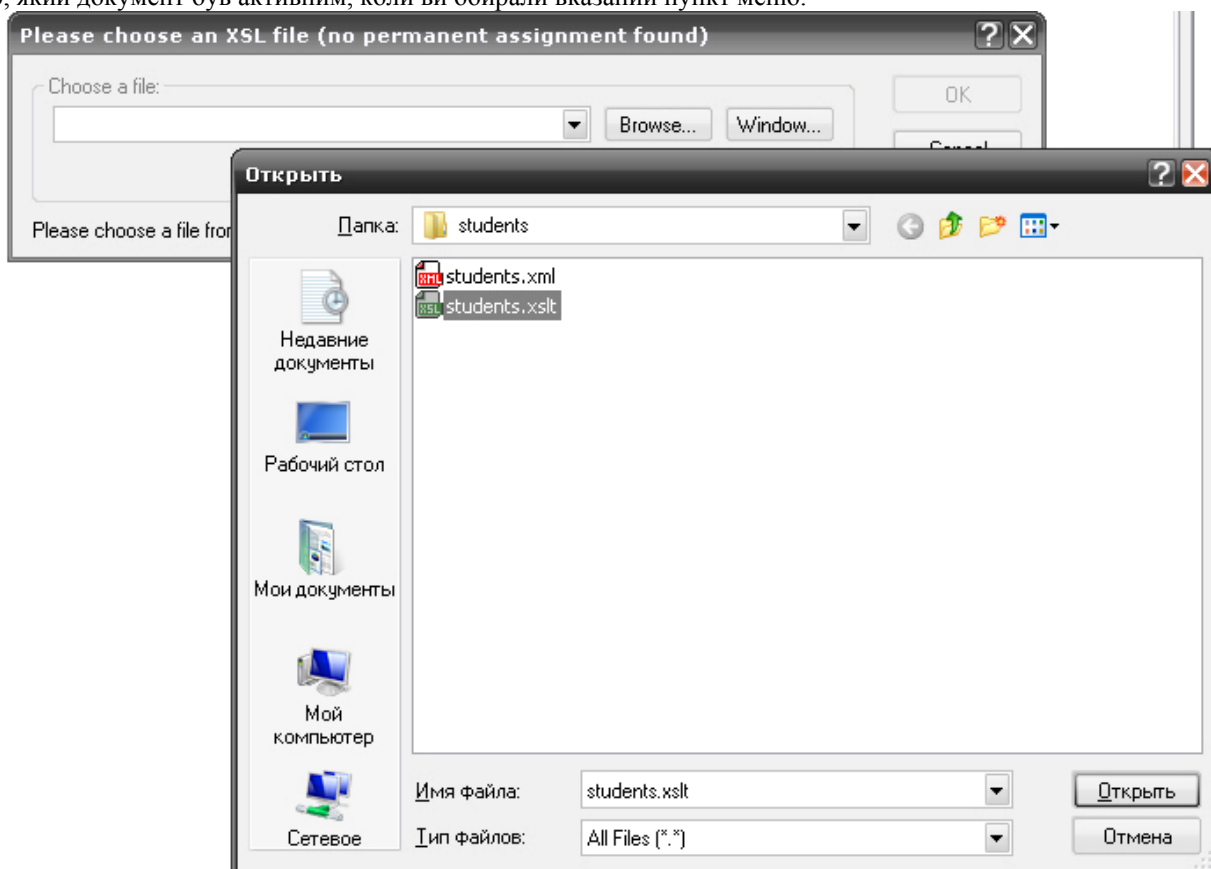
```

Як видно з коду, теги мови HTML записуються в таблиці стилів XSLT як звичайний текст. Процесор XSLT не обробляє їх, оскільки вони оголошені в іншому просторі імен. Ці теги перейдуть в конвертований документ без змін, але будуть записані згідно правил мови HTML/XHTML, оскільки в елементі `xsl:output` вказаний спосіб трансформації в HTML.

Після створення необхідних документів, необхідно застосувати створену таблицю стилів до вхідного XML-документу, обробити його та отримати на виході html-документ. Щоб здійснити таку трансформацію потрібно скористатись спеціальними програмами або скриптом, які призначені для конвертування документів. Через кілька пар ви самі можете написати такий простий скрипт, але зараз ми використаємо для цього редактор Altova XML Spy. Отже, обираємо пункт меню **XSL/XQuery->XSL Transformation** або скористатись гарячою функціональною клавішею **F10**.



Після цього з'явиться діалогове вікно, в якому потрібно обрати необхідний XML або XSLT документ, в залежності від того, який документ був активним, коли ви обирали вказаний пункт меню.





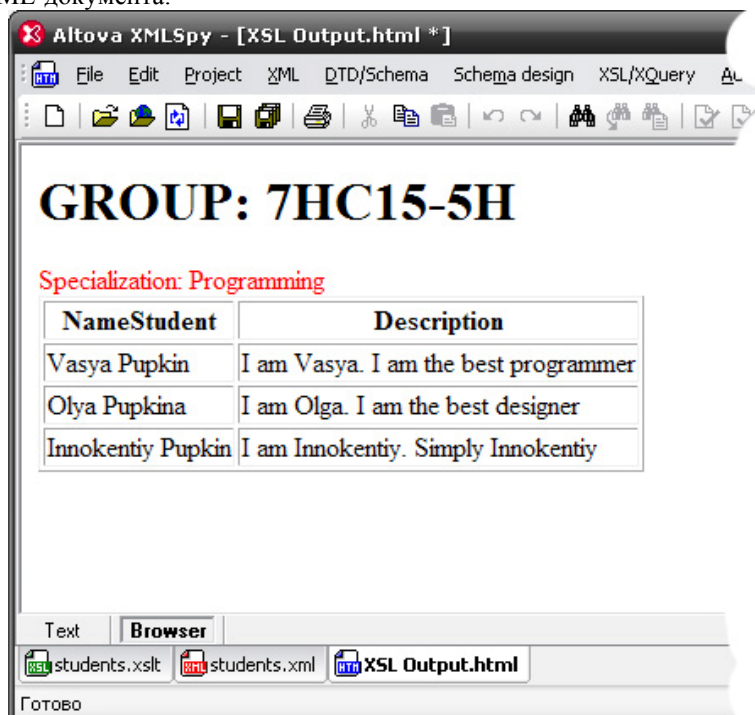
Доречі, якщо в XML документі одразу прописати шлях до XSLT документа, який їй відповідає, тоді постійно його обирати при кожному тесті не доведеться. Для визначення зв'язаної таблиці стилів використовується інструкція по обробці `<?xml-stylesheet?>`:

students.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" src="students.xslt" ?>

<group name="7HC15-5H" spec="Programming">
  <student name="Vasya Pupkin"> I am Vasya. I am the best programmer</student>
  <student name="Olya Pupkina"> I am Olga. I am the best designer</student>
  <student name="Innokentiy Pupkin"> I am Innokentiy. Simply Innokentiy</student>
</group>
```

Наступним кроком буде власне конвертування дерева XML-документа згідно обраної таблиці стилів XSLT і утворення HTML-документа.



Якщо перейти на закладку **Text**, то можна переглянути код новоутвореного HTML-документа:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>groups demo</title>
  </head>
  <body>
    <h1>GROUP: 7HC15-5H</h1><font color="#FF0000" size="5"><span>
      Specialization: Programming</span></font>
    <table border="1" style="bordercolor:#FF0000;">
      <tbody>
        <tr>
          <th>NameStudent</th>
          <th>Description</th>
        </tr>
        <tr>
          <td>Vasya Pupkin</td>
          <td> I am Vasya. I am the best programmer</td>
        </tr>
        <tr>
          <td>Olya Pupkina</td>
          <td> I am Olga. I am the best designer</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```



```

        <tr>
            <td>Innokentiy Pupkin</td>
            <td> I am Innokentiy. Simply Innokentiy</td>
        </tr>
    </tbody>
</table>
</body>
</html>

```

Як видно з вище представленого коду, ми отримали звичайну HTML-сторінку з даними, які були розміщені в XML-документі. Відступів відповідних тегів ми досягли за рахунок встановлення в елементі `xsl:output` атрибуту **indent** в значення **yes**.

3. Елементи XSLT

3.1. Рекурсивний виклик шаблонів. Елемент `xsl:apply-templates`

Та XSLT на цьому не закінчується. Як ми вже згадували вище, дана мова містить близько 3,5 десятків елементів, які також варті уваги. Розглянемо основні з них, тобто ті, які використовуються найчастіше.

Першим елементом, який на це заслуговує буде ще недорозглянутий нами елемент **`xsl:apply-templates`**. Даний елемент, як ми вже говорили застосовує на місці свого виклику інші шаблони, рекурсивно викликаючи їх та обробляючи. Частіше всього він записується всередині елемента `xsl:template` та не містить тіла. Але в даного елемента є ще два обов'язкових атрибута.

```

<xsl:apply-templates [select="вираз"]
                    [mode="Режим обробки"]>
    Конструктор послідовності
</xsl:apply-templates>

```

Атрибути **select** та **mode** мають аналогічне застосування, що і в елемента `xsl:template`. Але в даному випадку, атрибут **select** використовується скоріше для обмеження обробки лише вказаними в виразі вузлами.

Наприклад, необхідно обробити лише шаблони, які стосуються елемента `Student`:

```

<xsl:template match="/">
    <xsl:apply-templates select="/Student" />
</xsl:template>

```

3.2. Елементи `xsl:attribute` та `xsl:element`

Мова XSLT дозволяє не лише конвертувати вже існуючі вузли, але і створювати нові. Так, для створення вузлів-елементів та вузлів-атрибутів використовуються елементи **`xsl:element`** та **`xsl:attribute`** відповідно. Їх синтаксис практично однаковий:

```

<xsl:element name="ім'я"
            [namespace="простір імен"]
            [use-attribute-sets="імена"]>
    вміст елемента
</xsl:element>

<xsl:attribute name="ім'я"
              [namespace="простір імен"]>
    вміст атрибута
</xsl:attribute>

```

В обох цих елементів існує по одному обов'язковому атрибуту **name**, який і задає назву елемента або атрибута. Сам зміст записується у вигляді конструктора послідовності, тобто шаблону у їх тілі.

Необов'язковий атрибут **namespace** дозволяє вказати URL адресу простору імен новоствореного елемента або атрибута. А необов'язковий атрибут **use-attribute-sets** - задати список імен наборів атрибутів, які повинні бути включені в створюваний елемент.

Для прикладу розглянемо ситуацію, коли існує база даних книг в форматі XML. При цьому в XML документі наявний динамічно змінюваний вузол-атрибут **color** елемента **book**, який містить інформацію про колір обкладинки книги. Тобто елемент `book` має орієнтовно наступний вигляд:

```

<book color="red" />

```

Отже, наші книги різного кольору, і нам необхідно в результуючому документі вивести дані про книги у вигляді таблиці. При цьому необхідно, щоб поле з інформацією про певну книгу мало колір її обкладинки. Для цього значення HTML-тега `font` ми можемо задати одним з наступних способів.



```
<font color="{@color}"></font>
<font color="#{@color}"></font>
```

Або скористатись для цього елементом `xsl:attribute`:

```
<font>
  <xsl:attribute name="color">red</xsl:attribute>
  <xsl:attribute name="color">#<xsl:value-of select="@color"></xsl:attribute>
</font>
```

3.3. Умови та цикли. Елементи `xsl:if`, `xsl:for-each` та `xsl:choose`

В мові XSLT існує ряд операторів, які дозволяють здійснити умовну обробку даних (`xsl:if` та `xsl:choose`) та елемент циклу `xsl:for-each`.

Елемент `xsl:if` дозволяє створювати прості умови та запускає конструктор послідовності (тіло оператора) лише у випадку, якщо умова істинна. Сама умова записується в обов'язковому і єдиному атрибуті `test`.

```
<xsl:if test="умова">
  шаблон, який буде виконаний, якщо умова істинна
</xsl:if>
```

Як ви вже зрозуміли, вираз, записаний в атрибуті `test`, обраховується і приводиться до логічного типу. У випадку, якщо вираз умови повинен містити спеціальні символи (наприклад: `<`, `>`, `&` тощо), їх необхідно замінити відповідними символічними сутностями.

```
<xsl:if test="5 < 6" />    <!-- невірний запис -->
<xsl:if test="5 &lt; 6" />  <!-- вірний запис -->
```

Наприклад, існує список квітів у вхідному XML-документі:

```
<list active="Конвалія">
  <item>Ромашка</item>
  <item>Конвалія</item>
  <item>Троянда</item>
</list>
```

Необхідно, щоб у вихідному документі був також сформований список значень, причому обраним елементом був пункт, який вказаний в атрибуті `action`:

```
<xsl:template match="item">
  <option>
    <xsl:if test=". = ../@active">
      <xsl:attribute name="selected">selected</xsl:attribute>
    </xsl:if>
    <xsl:value-of select="." />
  </option>
</xsl:template>
```

В вихідному документі ми отримаємо наступний результат:

```
<option>Ромашка</option>
<option selected>Конвалія</option>
<option>Троянда</option>
```

Ще один приклад. Припустимо, існує список, який містить інформацію про працівників:

```
<employees>
  <employee position="tester">Ivan Ivanov</employee>
  <employee position="web-programmer">Vova Vovanov</employee>
  <employee position="tester">Petro Petrov</employee>
</employees>
```

Необхідно змінити шрифт відображення даних про працівника на зелений, якщо атрибут, що містить посаду має значення `tester`:

```
<xsl:template match="employees">
  <xsl:apply-templates />
</xsl:template>
```



```
<xsl:template match="employee">
  <p>
    <xsl:if test="./@position = 'tester'">
      <font color="green"><xsl:value-of select="." /></font>
    </xsl:if>
  </p>
</xsl:template>
```

Результат буде наступний:

```
<p><font color="green">Ivan Ivanov</font></p>
<p />
<p><font color="green">Petro Petrov</font></p>
```

Нажаль, елемент `xsl:if` в XSLT не реалізовує конструкцію `if..else`. У випадку, якщо таке необхідно використовується елемент, який здійснює вибір `xsl:choose`.

Елемент **`xsl:choose`** реалізує оператор вибору з множини значень та являється аналогом оператора `switch` в мові програмування C/C++. В даного елемента немає жодного параметра, але в його тілі записується один або кілька елементів **`xsl:when`** та один необов'язковий елемент **`xsl:otherwise`**.

При обробці елемента `xsl:choose` процесор послідовно обраховує логічні вирази, що розміщуються в атрибуті **`test`** елемента **`xsl:when`**. У випадку, якщо тестовий вираз першого елемента буде істинний, тоді виконується його конструктор послідовності. Інакше, якщо жоден тестовий вираз не справився (рівний `false`), тоді виконується конструктор послідовності оператора **`xsl:otherwise`**, який не містить атрибутів.

У випадку відсутності елемента `xsl:otherwise` і відсутності істинної умови, елемент `xsl:choose` поверне пустий рядок. Слід також відмітити, що в інструкції `xsl:choose` завжди виконується не більше одного варіанту.

Отже, повний синтаксис елемента `xsl:choose` має наступний вигляд:

```
<xsl:choose>
  <xsl:when test="умова">шаблон, який буде виконаний, якщо умова істинна</xsl:when>
  .....
  <xsl:otherwise>дія по замовчуванню</xsl:otherwise>
</xsl:choose>
```

Наприклад, необхідно перевірити день тижня:

```
<xsl:choose>
  <xsl:when test="@day=1">Понеділок</xsl:when>
  <xsl:when test="@day=2">Вівторок</xsl:when>
  <xsl:when test="@day=3">Середа</xsl:when>
  .....
  <xsl:otherwise>Такого дня тижня не існує</xsl:otherwise>
</xsl:choose>
```

Для здійснення циклічної обробки, тобто у випадку необхідності створення у вихідному документі повторюваних частин структури, використовується елемент **`xsl:for-each`**.

```
<xsl:for-each select="умова">
  шаблон, що буде виконуватись поки умова істинна
</xsl:for-each>
```

Даний елемент має єдиний та обов'язковий параметр **`select`**, що містить вираз, результатом обрахунку якого повинна бути послідовність вузлів. В результаті отримується цикл, який виконується стільки разів, скільки вузлів в наборі.

Наприклад, необхідно вивести інформацію про студентів певної групи, інформація про яких зберігається в елементах `student` вхідного XML-документа.

```
<table>
  <tbody>
    <xsl:for-each select="/group/student">
      <tr>
        <td><xsl:value-of select="." /></td>
      </tr>
    </xsl:for-each>
  </tbody>
</table>
```



3.4. Сортування. Елемент `xsl:sort`

Як було видно вище, при трансформації документа елементами `xsl:for-each` та `xsl:apply-templates`, обрані вузли по замовчуванню обробляються в порядку перегляду документа, який залежить від виразу, який використовується в атрибуті `select` цих елементів. Змінити цей порядок можна за допомогою сортування вузлів, яке здійснюється елементом `xsl:sort`.

Синтаксис даного елемента наступний:

```
<xsl:sort select="вираз"
  [case-order="lower-first | upper-first"]
  [data-type="number | text"]
  [lang="мова"]
  [order="descending | ascending"]
/>
```

Обов'язковий атрибут `select` дозволяє вказати ключовий вираз, який обраховується для кожного вузла обробляємого набору, конвертується в рядок і використовується як значення ключа при сортуванні. Значенням по замовчуванню даного атрибута являється (.), що означає, що в якості значення ключа для кожного вузла використовується його рядкове значення.

Після цього всі вузли набору сортуються по отриманим рядковим значенням своїх ключів. Доречі, елементів `xsl:sort` можна вказувати кілька і кожен з них буде задавати наступний ключ сортування.

Елемент `xsl:sort` містить також наступні **необов'язкові атрибути**:

- **case-order** – визначає порядок сортування символів різних регістрів, тобто заглавні чи прописні літери повинні йти першими. Значення по замовчуванню залежить від процесора та мови сортування, вказаної в атрибуті `lang`. Як правило, заглавні літери (`upper-first`) йдуть першими;
- **data-type** – вказує на різницю в сортуванні числових та текстових даних, тобто числові чи текстові дані будуть йти першими при сортуванні. Значення по замовчуванню для даного атрибута являється `text`;
- **lang** – вказує на те, яка мова при сортуванні буде грати основну роль. Як Ви знаєте ASCII коди символів різних мов відрізняються, тому і символи різних алфавітів можуть мати різний порядок сортування. Якщо значення даного атрибута не вказане, процесор може визначити мову, виходячи з системних налаштувань або взяти за основу коди символів Unicode;
- **order** – напрямок сортування: в зростаючому (`ascending`) чи спадаючому (`descending`) порядку сортувати вузли послідовності. Значенням по замовчуванню являється `ascending`.

В якості прикладу, здійснимо сортування студентів певної групи по імені в зростаючому порядку, а потім по прізвищу в спадаючому. Припустимо, що імена та прізвища задаються в якості атрибутів `name` та `surname` елементів `student` вхідного XML-документа.

```
<table>
  <xsl:for-each select="/group/student">
    <xsl:sort select="@name" />
    <xsl:sort select="@surname" order="descending"/>
    <tr>
      <td><xsl:value-of select="."/></td>
    </tr>
  </xsl:for-each>
</table>
```

Варто відмітити, що кожен процесор XSLT може по своєму інтерпретувати і виконувати сортування, тому на різних процесорах результат може відрізнятися.

3.5. Копіювання вузлів. Елементи `xsl:copy` та `xsl:copy-of`

Таблиця стилів може включати в себе не лише створення нових вузлів, але і копіювання існуючих. Для цього слід скористатись елементами `xsl:copy` та `xsl:copy-of`. Ці елементи дозволяють здійснювати копіювання вузлів, але кожен по різному.

Елемент `xsl:copy` здійснює просте копіювання контекстного вузла на місці виклику. Тобто створює копію поточного вузла, не залежно від його типу. При цьому дочірні вузли і вузли атрибутів в вихідний документ не копіюються.

Наприклад:

```
<xsl:copy>
  <xsl:apply-templates select="@* | node()" />
</xsl:copy>
```

У випадку розміщення даного елемента в шаблон, у вихідний документ буде скопійований весь вміст вказаного контекстного вузла.



Елемент **xsl:copy-of** містить атрибут **select**, в якому вказується умова копіювання. Як правило, результатом роботи даного елемента являється рядкове значення, але не завжди. Якщо результатом обрахунку являється набір вузлів, тоді він копіюється в вихідний документ разом з своїми потомками.

Наприклад:

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

3.6. Коментарі та текстові вузли

Шаблон конвертування може містити текстові вузли, які при виконанні самого шаблону будуть скопійовані в результуюче дерево документа. Отже, для того, щоб у результуючому документі вивести звичайний текст, його необхідно задати в шаблоні XSLT.

Створювати текстові вузли можна **двома способами**:

1. Використовуючи літерал з значенням необхідного текстового вузла. Наприклад:

```
<font color="#ff0000">
  Specialization: <xsl:value-of select="@spec" />
</font>
```

2. Використовуючи елемент **xsl:text**:

```
<xsl:text [disable-output-escaping="yes | no"]>
  вміст тестового вузла
</xsl:text>
```

Наприклад,

```
<font color="#ff0000">
  <xsl:text>Specialization: <xsl:value-of select="@spec" /></xsl:text>
</font>
<!-- або -->
<font color="#ff0000">
  <xsl:text>Specialization: </xsl:text><xsl:value-of select="@spec" />
</font>
```

Варто відмітити, що при створенні текстових вузлів за допомогою елемента **xsl:text** пропуски та спеціальні символи будуть збережені, на відміну від використання першого варіанту. Якщо це не потрібно, тобто ви бажаєте, щоб спеціальні символи були замінені відповідними символічними або вбудованими сутностями, то в необов'язковому атрибуті **disable-output-escaping** варто вказати значення **yes**. Як ви вже зрозуміли, по замовчуванню значення даного атрибута рівне **no**.

Для створення у вихідному документі коментаря вигляду

```
<!-- текст коментаря -->
```

в XSLT використовується елемент **xsl:comment**

```
<xsl:comment>текст коментаря</xsl:comment>
```

Текст коментаря повинен містити лише текстові вузли, оскільки вузли інших типів в результаті трансформації будуть або проігноровані, або викличуть помилку. Також не варто забувати, що в тілі коментаря не можна використовувати більше символів (-) підряд.

Наприклад,

```
<xsl:comment>Тут закоментовані секретні дані</xsl:comment>
```

Шаблон, що містить такий рядок створить наступний коментар в вихідному документі:

```
<!-- Тут закоментовані секретні дані -->
```

3.7. Створення інструкцій по обробці

Для створення в вихідному документі вузла інструкції по обробці, слід скористатись елементом **xsl:processing-instruction**:

```
<xsl:processing-instruction name="ім'я">
  вміст (набір атрибутів)
</xsl:processing-instruction>
```



Обов'язковий атрибут `name` дозволяє вказати ім'я нової інструкції по обробці, тобто назву додатку, якому буде адресована дана інструкція. Вміст самого елемента вказує на набір атрибутів і їх значень, що містить інструкція по обробці. Він повинен містити лише текстові вузли, інакше процесор XSLT може згенерувати помилку або проігнорувати нетекстові вузли разом з їх вмістом.

Слід зауважити, що XML-декларація не може формуватись даним елементом, оскільки вона не являється інструкцією по обробці, хоча і схожа з нею синтаксично. Для цього використовується елемент `xsl:output` таблиці стилів.

Наприклад, для створення інструкції по обробці `<?xml-stylesheet type="text/xsl" href="styles.xml"?>` потрібно прописати наступне:

```
<xsl:processing-instruction name="xml-stylesheet">
  <xsl:text>type="text/xsl" href="styles.xml"</xsl:text>
</xsl:processing-instruction>

<!-- або -->
<xsl:processing-instruction name="xml-stylesheet">
  type="text/xsl" href="styles.xml"
</xsl:processing-instruction>
```

4. Змінні в XSLT

В XSLT можна використовувати змінні, які мають імена та з якими зв'язується певне значення. Оголошуються змінні за допомогою елемента `<xsl:variable>` одним з наступних способів:

```
<xsl:variable name="ім'я" select="значення" [as="тип"] />
<xsl:variable name="ім'я" [as="тип"]>значення</xsl:variable>

<!-- за допомогою конструктора послідовності -->
<xsl:variable name="ім'я" [as="тип"]>
  <xsl:value-of select="вираз XPath" />
</xsl:variable>
```

Атрибут `name` дозволяє задати розширене QName ім'я майбутньої змінної, в атрибуті `select` задається значення змінної, яке може бути як звичайним значенням, так і виразом XPath. Необов'язковий атрибут `as` дозволяє визначити тип об'єкта.

Наприклад,

```
<xsl:variable name="url" select="http://www.google.com" />
<xsl:variable name="font_size" as="xs:integer">12 to 50</xsl:variable>
<xsl:variable name="producer" select="./Item/*" />

<xsl:variable name="countProducts" as="xs:integer">
  <xsl:value-of select="count(//products)" />
</xsl:variable>
```

В першому випадку створиться змінна з іменем `url`, яка містить рядкове значення ["http://www.google.com"](http://www.google.com). В другому - змінна з іменем `font_size`, яка містить послідовність значень типу `integer` від 12 до 50. В третьому - змінна з іменем `producer`, яка буде містити значення, що буде обраховано виразом `"./Item/*"`, тобто посилання на всі дочірні елементи елемента `Item`, який розташований після поточного контекстного вузла. Четверта змінна з іменем `countProducts` буде містити цілочисельне значення кількості елементів `products`, які являються потомками поточного контекстного вузла.

Якщо об'єкт не отриманий з атрибута `select` або з елемента `xsl:variable`, то ім'я змінної по замовчуванню зв'язується з пустим рядком.

Для того, щоб використати змінну потрібно вказати її ім'я з знаком `$` на початку. Наприклад, щоб звернутись до змінної `url` потрібно написати `$url`. Це зроблено для того, щоб відрізнити змінні від шляхів вибірки.

Наприклад,

```
<xsl:value-of select="concat('Welcome to ', $url)" />
<xsl:value-of select="$producer/@name" />
```

Слід відмітити, що у випадку використання змінної в якості значень елементів HTML/XHTML, їх ім'я потрібно вказувати в фігурних дужках. Наприклад:

```
<!-- існує змінна, що містить значення кольору -->
<xsl:variable name="color">#00FF70</xsl:variable>

<!-- в якості кольору фону тіла сторінки використовується значення змінної color -->
<body bgcolor="{ $color }">
</body>
```



Вищеописаний варіант використання змінних може повноцінно замінити використання елемента `xsl:attribute`.

Кожна змінна характеризується областю бачення, яка визначає область її дії. В залежності від порядку створення, змінна може мати глобальну (видима в межах всього документа XSLT) або локальну (видима лише в межах свого батьківського елемента) область бачення. Говорячи іншими словами, змінні, оголошені безпосередньо в кореневому елементі `xsl:stylesheet` (`xsl:transform`) являються глобальними змінними, всі інші – локальні.

Варто відмітити, що використання змінних в XSLT відрізняється від їх використання в мовах програмування типу C++, Java, C# тощо, в зв'язку з тим, що їх значення не може бути змінено після присвоєння. Фактично, змінна XSLT являється лише об'єктом, який зручно використовувати, коли певним значенням потрібно користуватись в кількох місцях документа, а його розрахунок громіздкий або складний.

5. Іменовані шаблони та параметри

На самому початку нашого уроку ми говорили про елемент `xsl:template`, який дозволяє задати шаблон трансформації. Замість того, щоб за допомогою його атрибута `match` вказувати, яка частина вхідного документа повинна бути трансформована, шаблону можна надати ім'я і викликати незалежно від контексту трансформації. Нагадаємо, що ім'я шаблону задається за допомогою атрибута `name`.

Після того, як шаблону буде присвоєне ім'я він стане **іменованим**. Іменовані шаблони по своїй суті дуже схожі з процедурами та функціями в мовах програмування, їм навіть можна передавати параметри. Для передачі параметрів використовується елемент `xsl:param`, який може приймати дві форми запису:

```
<xsl:param name="ім'я" select="вираз-значення" />
<xsl:param name="ім'я">вираз-значення</xsl:param>
```

В обов'язковому атрибуті `name` вказується ім'я параметра, а його значення можна вказувати або в атрибуті `select` або в тілі елемента.

Наприклад,

```
<xsl:template name="myTempl">
  <xsl:param name="a" select="12" />
  <xsl:param name="b">22</xsl:param>
  <xsl:param name="c" />

  <xsl:if test="$a > 10">
    .....
  </xsl:if>
</xsl:template>
```

Виклик іменованого шаблону здійснюється за допомогою елемента `xsl:call-template`.

```
<xsl:call-template name="ім'я шаблону" />
```

Наприклад:

```
<xsl:call-template name="myTempl"></xsl:call-template>
```

Якщо в іменованій шаблон при виклику потрібно передати параметр, то для цього в тілі елемента `xsl:call-template` потрібно викликати елемент `xsl:with-param`, який має синтаксис аналогічний елементу `xsl:param`.

Тобто для виклику нашого іменованого шаблону слід написати наступне:

```
<xsl:call-template name="myTempl">
  <xsl:with-param name="a" select="10" />
  <xsl:with-param name="b">17</xsl:with-param>
</xsl:call-template>
```

6. Домашнє завдання

1. Написати таблицю стилів XSLT для трансформації XML документа «Академія», розробленого раніше (урок 3) в HTML-документ.
2. Написати таблицю стилів XSLT для XML документа [Object.xml](#) та конвертувати його в HTML-документ. Новоутворена Web-сторінка повинна мати орієнтовний вигляд сторінки MSDN, яку можна переглянути за адресою: [Object - клас](#).



Або ниже:

Object - класс

.NET Framework 4

Другие версии

2 из 2 оценили этот материал как полезный

Оценить эту тему



Поддерживает все классы в иерархии классов .NET Framework и предоставляет низкоуровневые службы для производных классов. Он является исходным базовым классом для всех классов платформы .NET Framework и корнем иерархии типов.

Иерархия наследования

System.Object

Все классы, структуры, перечисления и делегаты.

Пространство имен: **System**

Сборка: mscorlib (в mscorlib.dll)

Синтаксис

```
C# C++ F# VB
[SerializableAttribute]
[ClassInterfaceAttribute(ClassInterfaceType.AutoDual)]
[ComVisibleAttribute(true)]
public class Object
```

Тип **Object** предоставляет следующие члены.

Конструкторы



	Имя	Описание
	Object	Инициализирует новый экземпляр класса Object .

[В начало страницы](#)

Методы

	Имя	Описание
	Equals(Object)	Определяет, равен ли заданный объект Object текущему объекту Object .
	Equals(Object, Object)	Определяет, считаются ли равными указанные экземпляры объектов.
	Finalize	Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки, перед тем как объект будет утилизирован в процессе сборки мусора.
	GetHashCode	Играет роль хэш-функции для определенного типа.
	GetType	Возвращает объект Type для текущего экземпляра.
	MemberwiseClone	Создает неполную копию текущего объекта Object .
	ReferenceEquals	Определяет, совпадают ли указанные экземпляры Object .



	ReferenceEquals	Определяет, совпадают ли указанные экземпляры Object .
	ToString	Возвращение строки, представляющей текущий объект.

[В начало страницы](#)

▲ Заметки

Обычно в языках программирования не требуется объявлять класс наследником **Object**, так как наследование происходит неявно.

Так как все классы в платформе .NET Framework являются производными класса **Object**, все методы, определенные в классе **Object**, доступны для всех объектов в системе. В производных классах некоторые из этих методов, включая перечисленные ниже, могут переопределяться и переопределяются:

- [Equals](#) — поддерживает сравнение объектов.
- [Finalize](#) — выполняет операции очистки перед автоматической утилизацией объекта.
- [GetHashCode](#) — создает число, соответствующее значению объекта, обеспечивающего возможность использования хэш-таблицы.
- [ToString](#) — создает понятную для пользователя строку текста, в которой описывается экземпляр класса.

Особенности производительности

При проектировании таких классов, как коллекция, в которой должны обрабатываться любые типы объектов, можно создать члены класса, принимающие экземпляр класса **Object**. Однако процесс упаковки-преобразования и распаковки-преобразования типа приводит к снижению производительности. Если известно, что новый класс будет часто обрабатывать значения определенных типов, для минимизации снижения производительности в результате упаковки-преобразования можно прибегнуть к одной из двух тактик.

- Первая тактика предполагает создание общего метода, принимающего тип **Object** и задание особых перегрузок метода для конкретных типов, принимающих значения всех типов, которые предположительно будут обрабатываться классом. При наличии особого метода для определенного типа, принимающего тип параметра вызова, вызывается этот особый метод, и упаковка-преобразование не происходит. Если у метода нет аргумента, совпадающего по типу с параметром вызова, вызывается общий метод и выполняется упаковка-преобразование.
- Вторая тактика заключается в проектировании класса и его методов как универсальных. Среда CLR при создании экземпляра класса создает закрытый универсальный тип и задает для него аргумент. Универсальный метод задается для определенного типа и его можно вызывать без упаковки-преобразования и параметра вызова.

Хотя иногда нужно разрабатывать именно классы общего назначения, принимающие и возвращающие типы **Object**, для повышения производительности можно также создать особые классы для конкретных часто используемых типов. Например, создав особый класс, задающий и возвращающий логические значения, можно избежать затрат на упаковку-преобразование и распаковку-преобразование значений такого типа.

► Примеры

▲ Сведения о версии

.NET Framework

Поддерживается в версиях: 4, 3.5, 3.0, 2.0, 1.1, 1.0

.NET Framework (клиентский профиль)

Примітки!

1. Каждое домашнее задание оценивается отдельно, а оценка зависит от реализации.
2. Другое задание (класс **Object**) может быть оценено как курсовая работа по курсу XML.