



Урок 7

План заняття:

1. Аналіз XML документів
 - 1.1. Введення
 - 1.2. SAX-парсери
 - 1.3. DOM-парсери
 - 1.4. Переваги і недоліки моделей SAX і DOM
2. Основи об'єктної моделі XML документа (XML DOM). Властивості, методи та події DOM API
3. Приклад використання DOM API
4. Відловлення синтаксичних помилок
5. Обхід дерева DOM
6. Зміна структури XML документа
7. Відбір вузлів за допомогою XPath. XSLT трансформація засобами DOM
8. Робота з DOM XML в інших браузерах
9. Домашнє завдання

1. Аналіз XML документів

1.1. Введення

На попередніх парах ми часто говорили з вами про програми-обробники (процесори) XML документів. Нагадаємо, що вони читають XML документи, аналізують їх, а також у випадку необхідності дозволяють маніпулювати їх вмістом (витягують потрібну інформацію, конвертують документи тощо). Для здійснення всіх цих дій по взаємодії додатків і XML-документів, використовується стандартний набір властивостей, подій і методів, тобто API. Процесори XML власне і використовують цей API для доступу і роботи з XML документами.

Отже, знаючи бібліотеку функцій або класів для роботи з XML та будь-яку мову програмування або мову скриптів, ви можете з легкістю написати свій власний процесор для обробки XML документів.

Сам процес аналізу XML документа проходить **2 етапи**:

1. **Лексичний аналіз або сканування XML-документа**, який здійснюють спеціальні програми – **сканери (scanners)**. На даному етапі документ розбивається на окремі неподільні елементи (токени - tokens), якими являються теги, роздільники (сепаратори), текстові лексеми, службові слова. Також здійснюється перевірка їх зв'язків.
2. **Граматичний аналіз**, в результаті якого аналізується логічна структура документа. Елементи, отримані в результаті лексичного аналізу, збираються у вирази, які об'єднуються в блоки, а блоки – в модулі (абзаци, пункти, параграфи). Граматичний аналіз здійснюють **програми-аналізатори**, тобто **парсери**.

Парсерів XML на сьогоднішній день існує дуже багато. Але всі їх можна поділити на **2 основні групи**:

1. аналізатори, які генерують дерево елементів XML (tree-based parsing) - DOM парсери;
2. аналізатори, основані на подіях (event-based parsing) - SAX парсери.

DOM і SAX аналізатори призначені для доступу до різних частин документа, зокрема до елементів XML-документів. По суті, це набір рекомендацій щодо того, як і через які інтерфейси програма-клієнт буде використовувати структуру документа. Тому не дивлячись на відмінності обох підходів, основні терміни в них спільні.

1.2. SAX-парсери

SAX (Simple API for XML) – це програмний інтерфейс додатків, який здійснює послідовне читання/запис XML-документів. Розробником SAX-парсера вважається Девід Меггінсон, хоча первинно її розробляли кілька програмістів з maillist'a xml-dev. Меггінсон лише об'єднав всі наробки в єдине ціле і першим написав код для SAX. Варто відмітити, що SAX не являється стандартом W3C і основний сайт проекту знаходиться по адресі <http://www.saxproject.org>.

Зазвичай SAX-парсери потребують фіксованої кількості пам'яті для своєї роботи і не дозволяють змінювати вміст документа: додавати нові елементи, або змінювати вже існуючі. Все, що вони роблять – це читають XML документ і при появі відкриваючого або закриваючого тега, текстових даних, помилок тощо повідомляють про це викликаючий додаток. Кожна така поява вважається подією, яка викликає відповідну функцію обробник (startElement(), endElement(), characters(), processingInstruction() тощо).

SAX-парсер на сьогоднішній день являється фактичним стандартом аналізаторів, основаних на подіях. Нині використовується друга версія цього стандарту – **SAX2**, який входить до складу багатьох XML аналізаторів, наприклад, Xerces2. Доречі, до нового стандарту SAX2 доданий інтерфейс **IMXWriter**, який дозволяє здійснювати модифікацію XML документів.

Реалізації SAX-парсерів можуть відрізнятися одна від одної, але в цілому всі вони працюють однотипно. Зазвичай, такі парсери застосовуються для швидкого пошуку по XML-документу або для читання XML-потоків великого об'єму.



1.3. DOM-парсери

Другим стандартом для обробки XML-даних являється DOM. **DOM (Document Object Model, об'єктна модель документа)** – це незалежний від платформи і мови програмний інтерфейс, який дозволяє програмам і скриптам отримувати доступ до вмісту документів та маніпулювати ним: змінювати вміст, структуру, оформлення тощо. По суті, DOM являє собою стандартний набір об'єктів та методів для роботи з вмістом (X)HTML та XML документів.

На відміну від SAX, ці парсери дозволяють здійснювати довільні операції з XML-даними в зручній формі - представляючи XML-документ у вигляді дерева вузлів. Кожен вузол дерева являє собою елемент, атрибут, текстовий вузол або інший об'єкт. Самі вузли з'єднані між собою відносинами батьківський-дочірний.

Поточною версією специфікації DOM являється DOM Level 3, але деякі частини даної специфікації підтримуються дуже слабо. Рекомендованими консорціумом являються DOM Level 2 та лише деякі частини DOM Level 3. Приведемо короткі **характеристики специфікацій**, оскільки на курсі JavaScript ви вже вивчили основні їх відмінності:

- **DOM Level 0** (1997 р.) – включає в себе стандартизацію об'єктної моделі для Web-браузерів.
- **DOM Level 1** (1998 р.) – описує базові функціональні можливості DOM для XML 1.0 і HTML 4.0: отримання дерева вузлів документа, можливість зміни і додавання даних тощо.
- **DOM Level 2** (2000 р.) – додана підтримка каскадних таблиць стилів (CSS), просторів імен, визначені стандартні події користувацького інтерфейсу, а також покращені методи маніпуляції деревовидною структурою документа.
- **DOM Level 3** (2004 р.) – підтримка користувацьких подій, а також механізму для збереження та зчитування документів. Даний рівень складається з шести різних специфікацій, які являються додатковими розширеннями DOM:
 - DOM Level 3 Core;
 - DOM Level 3 Load and Save;
 - DOM Level 3 XPath;
 - DOM Level 3 Views and Formatting;
 - DOM Level 3 Requirements;
 - DOM Level 3 Validation.
- **DOM Level 4** (квітень 2012 р.) – вийшла чергова версія **робочого проекту**, яка внесла деякі корективи в попередній стандарт, зокрема вдосконалена робота з помилками (exception), додані нові колекції тощо.

Нагадаємо також, що консорціум W3C розділяє концепції DOM на **5 категорій**: базовий DOM (ядро DOM), HTML DOM, DOM CSS, DOM Events (події DOM) та DOM XML.

Отже, подібно SAX-парсерам, модель DOM також підтримує генерацію подій, хоч і частково. Наприклад, парсер MSXML 4.0 генерує події:

- Event ondataavailable;
- Event onreadystatechange;
- Event ontransformnode.

Щодо швидкості обробки, то DOM-парсери значно повільніші за SAX-парсери, оскільки завантажують в пам'ять повністю весь документ. Але це дозволяє здійснювати доступ до будь-якого об'єкта документа, на відміну від SAX, який дозволяє лише послідовно зчитати дані.

1.4. Переваги і недоліки моделей SAX і DOM

Кожен з вищеописаних типів синтаксичних аналізаторів (парсерів) має свої **переваги і недоліки**. Для кращого сприйняття відобразимо їх у вигляді співставної таблиці, де недоліки виділимо коричневим кольором та курсивом.

DOM-парсери	SAX-парсери
Дозволяє здійснити вибіркоковий (random access) доступ до елементів документа	<i>Доступ лише послідовний, дозволяють переглянути XML-документ лише один раз від початку до кінця, і не можуть повернутись назад</i>
Існує багато реалізацій DOM-парсерів і під різноманітні платформи	<i>В зв'язку з непопулярністю, існує небагато реалізацій SAX-парсерів</i>
Дозволяє модифікувати дані XML-документа і зберігати їх	<i>Використовується лише для читання XML-документів</i>
Підтримує XSLT-трансформацію	-
Дозволяє виявити помилки структури документа до етапу роботи з документом. В результаті в програму передається інформація у вигляді виключень або від'ємного результату дій	Помилки виявляються лише на місці їх появи. Обробка помилок відбувається через методи інтерфейсу ErrorHandler, які визначаються програмою-клієнтом
<i>Повільні і потребують більше пам'яті, в порівнянні з SAX, оскільки завантажують в пам'ять весь XML-документ</i>	Швидкі і потребують менше пам'яті для обробки XML-документів, оскільки в пам'яті існує лише невелика частина документа
<i>Процес обробки (парсинг) XML-документа неконтролюємий</i>	Можна зупинити процес обробки XML документа



	(парсинг) в будь-який момент, що дозволяє використовувати SAX-парсери для пошуку унікальних даних. Наприклад, потрібно знайти конкретний продукт в каталозі товарів і зупинити пошук.
<i>Зчитує весь вміст XML-документа і тому неефективний при мережевому доступу до даних</i>	Дозволяє зчитувати дані порціями, чим знижується ресурсоемкість процесу обробки даних. Тобто можна здійснювати частковий парсинг документа

Отже, DOM слід використовувати в **наступних випадках**:

- необхідна XSLT-трансформація;
- необхідний доступ за допомогою XPath;
- необхідно здійснювати модифікацію XML документа та записувати документи (SAX не може записувати документи);
- необхідний вибіркового доступ до даних документа.

Оскільки SAX-парсери обробляють дані поступово, по мірі їх зчитування, а не завантажують в пам'ять весь XML-документ, як це роблять DOM-парсери, то зчитування з їх допомогою буде відбуватись швидше. Тому якщо необхідне звичайне зчитування даних XML, слід скористатись засобами SAX-парсерів. Але така ситуація трапляється досить рідко, тобто лише одного читання для роботи недостатньо, тому ми зробимо основний акцент на вивчення синтаксичних аналізаторів DOM XML.

2. Основи об'єктної моделі XML документа (DOM XML). Властивості, методи та події об'єктів DOM API

По суті, DOM являє собою набір класів для представлення XML документів. Інтерфейси DOM API написані на мові IDL (Interface Definition Language), яка описує незалежні від мови програмування інтерфейси. Тим не менше, інтерфейси DOM API можна реалізовувати на довільній об'єктно-орієнтованій мові програмування. На сьогоднішній день існують реалізації на мовах C++, Java, JavaScript, Python, Perl, PHP, C# тощо. Існує також багато DOM парсерів від різних фірм-розробників, зокрема від Oracle (вважаються найкращими для Java по багатьом параметрам) та Microsoft (найпопулярніші парсери для платформи Windows).

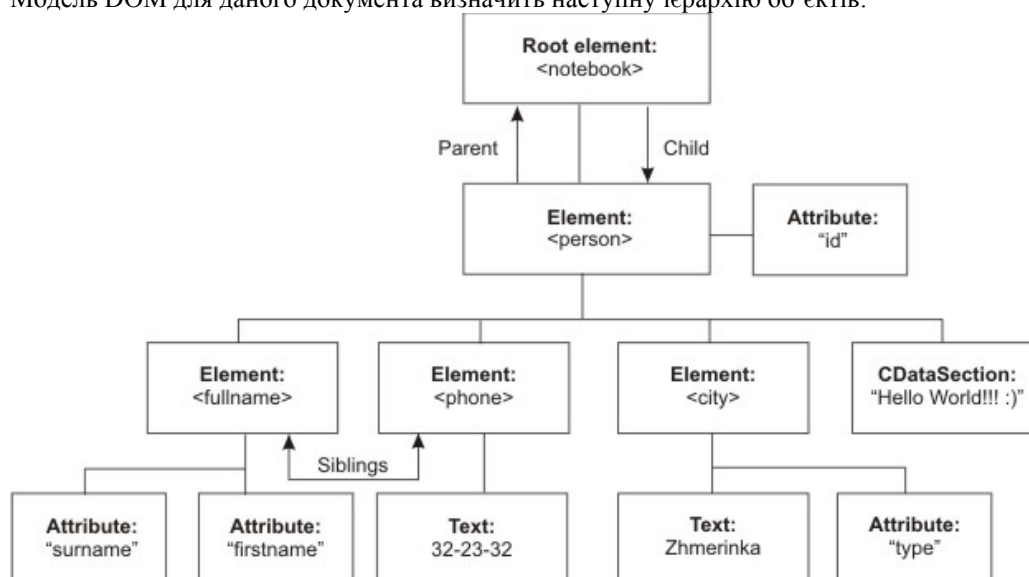
Перейдемо до практики. Як вже було сьогодні сказано, DOM-парсери розглядають структуру XML-документ у вигляді дерева. Наприклад, існує XML-документ «Записна книга», який має наступний вигляд:

notebook.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<notebook>
  <person id="P1">
    <fullname firstname="Vasja" surname="Pupkin"/>
    <phone>22-22-22</phone>
    <city type="сmt">Zhmerinka</city>
  </person>
  <person id="P2">
    <fullname firstname="Dasha" surname="Ivanova"/>
    <phone>333-33-33</phone>
    <city type="м">Kyev</city>
  </person>
  <![CDATA[ "Hello, World!!! :-)" ]]>
</notebook>
```



Модель DOM для даного документа визначить наступну ієрархію об'єктів:



Для доступу до кожного вузла дерева, а також для маніпуляції цими вузлами використовується набір об'єктів DOM API, які включають в себе необхідні методи, властивості та події. Ці об'єкти DOM XML часто називають вузлами. Приведемо їх список:

Об'єкт DOM	Опис
Node	Базовий тип для інших об'єктів
Document	Кореневий вузол документа (весь XML-документ)
Element	Вузол елемента
Attr або Attribute	Атрибут, який представлений парою «ім'я-значення»
Text	Текстова нода (вузол), який представляє текст, що міститься в елементі або атрибуті
CDATASection	Розділ CDATA
Comment	Коментар
Notation	Нотація, яка міститься в DTD-документі або в схемі
Entity	Сутність
Entity Reference	Посилання на сутність, яка має наступний вигляд: &Entity;
ProcessingInstruction	Інструкція по обробці
DocumentType	Оголошення типу документа (DTD)
Document Fragment	Фрагмент (частина) документу

Отже, дерево DOM складається з вузлів, типи та спільні властивості яких описані об'єктом **Node**. Все дерево описується об'єктом **Document**, елементи об'єктом **Element**, а їх атрибути об'єктом **Attribute**. Ці чотири об'єкти найчастіше використовуються, тому варто їх запам'ятати.

Всі вузли мають спільні і специфічні для кожного властивості і методи. Крім того, цей набір може дещо відрізнитись в кожному окремому парсері. Для демонстрації роботи DOM API для XML ми спочатку скористаємось парсером від фірми Microsoft MSXML, остання версія якого MSXML 6.0. Отже, саме його набір методів і властивостей ми розглянемо.

3. Приклад використання DOM API

Для демонстрації роботи DOM XML ми скористаємось парсером від фірми Microsoft MSXML, остання версія якого на момент виходу уроку MSXML 6.0. Парсер Microsoft XML – це COM-компонент (точніше ActiveX), який поставляється разом з Internet Explorer 5.0 та вище. В якості мови додатку, який буде здійснювати звернення до XML документів ми оберемо мову JavaScript, яку ви вже знаєте.

Центральним об'єктом XML DOM для Internet Explorer являється ActiveX-об'єкт **XMLDOM** або **DOMDocument**. Він фактично представляє собою кореневий елемент. Але він не єдиний. Для роботи з DOM API для XML використовуються і інші об'єкти. Наприклад, **DOMParser**, **XSLTProcessor**, **XMLSerializer**, **XMLHttpRequest** для **Firefox**, **Opera** та інших браузерів.

Для створення екземпляра об'єкта **DOMDocument** необхідно скористатись оператором **new**. Наприклад:

```
var obj = new ActiveXObject("ім'я об'єкта");
```



```
var obj = new ActiveXObject("Microsoft.XMLDOM");
або
var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
```

Елемент **documentElement** являється верхнім рівнем дерева XML. Цей елемент має один або кілька дочірніх елементів **childNodes**, які вказують на гілки дерева. Саме модель інтерфейса на основі вузлів використовується для отримання доступу до окремих елементів дерева.

Інші браузері використовують методи **createObject()** або **createDocument()**, які здійснюють аналогічні дії. Але при використанні даних методів для створення об'єктів існують деякі особливості, вірніше особливість. Про неї детальніше розказано в [розділі 8 «Робота з DOM XML в інших браузерах»](#) даного уроку.

```
var doc = CreateObject("Microsoft.XMLDOM");
```

Отже, в ActiveX-об'єкт для роботи з XML-документами можна створити одним з двох способів:

```
var doc = new ActiveXObject("MSXML2.DOMDocument.6.0"); //для Internet Explorer
var doc = CreateObject("Microsoft.XMLDOM");           //для інших браузерів
```

Після цього потрібно налаштувати даний об'єкт для подальшої роботи, встановивши **наступні властивості**:

- **async** - вказує на можливість асинхронного завантаження. Якщо він рівний false, синтаксичний аналізатор завершує виконання довільних дій до повного завантаження документа в пам'ять
- **validateOnParse** - вказує на необхідність перевірки документа (у відповідності до схеми або DTD документа) при його обробці
- **resolveExternals** - дозволити чи ні простори імен, DTD-документи, схеми чи зовнішні посилання під час обробки XML-документа

Код ініціалізації об'єкта DOM XML буде виглядати так:

```
//створюємо екземпляр парсера MSXML
var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
//var doc = CreateObject("Microsoft.XMLDOM");

doc.async = false;
doc.validateOnParse = false;
doc.resolveExternals = false;
```

До складу XML DOM входять методи та властивості для роботи з вмістом XML-документа та по обходу самого дерева. Ряд з них ви вже знаєте з курсу JavaScript (наприклад, **createElement**, **appendChild**, **hasChildNodes**, властивості **nodeName**, **nodeType**, колекції **attributes**, **childNodes** тощо), але є і специфічні для окремого парсера XML. Наприклад, MSXML містить такі методи як **load** (завантажує xml документ), **loadXML** (завантажує xml документ у вигляді тексту), **save** (зберігає xml документ), **error** (текст помилки).

Для початку напишемо простий скрипт, який дозволить доступатись до певних вузлів завантаженого XML-документа «Записна книга».

TestDOM.html

```
<html>
<head>
  <title>Test DOM XML</title>
  <script type="text/javascript">
    <!--
      //створюємо екземпляр парсера MSXML
      var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
      //var doc = CreateObject("Microsoft.XMLDOM");

      doc.async = false;
      doc.validateOnParse = false;
      doc.resolveExternals = false;
    <!-->
  </script>
</head>
<body>
```

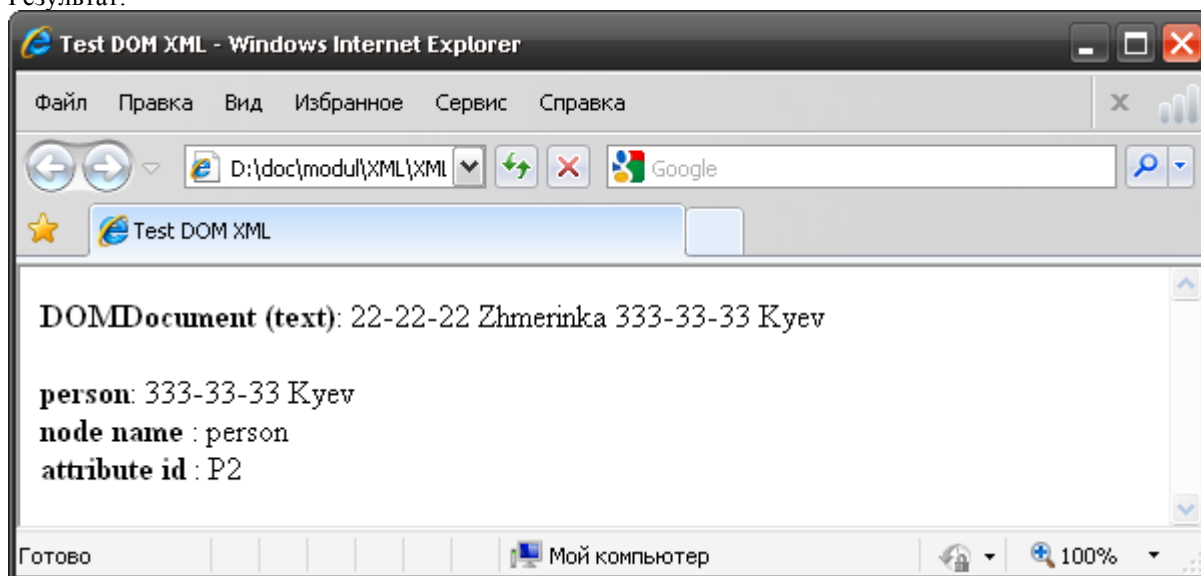


```
<script type="text/javascript">
<!--
    //завантажуємо XML документ для обробки
    if(!doc.load("notebook.xml")) {
        alert("Error loading file!");
    }

    //виводимо текстовий вміст завантаженого XML документа
    document.write("<b>DOMDocument (text)</b>: " + doc.text + "<br /><br />");
    //доступаємось до елемента <person> та виводимо його текстовий вміст
    document.write("<b>person</b>: " +
        doc.getElementsByTagName("person").item(1).text);

    var node = doc.getElementsByTagName("person").item(1);
    document.write("<b>node name</b> : " + node.nodeName + "<br />");
    document.write("<b>attribute id</b>: " + node.getAttribute("id"));
-->
</script>
</body>
</html>
```

Результат:



Властивість **text** дозволяє показати текстовий вміст певного вузла. Оскільки об'єкт **doc** містить повний екземпляр завантаженого XML документа, за допомогою властивості **text** можна вивести текстові дані всіх дочірніх елементів.

Для доступу до окремо взятого елемента XML по імені, слід скористатись методом **getElementsByTagName()**.

Щоб доступитись до окремого поля з колекції, наприклад, до першого елемента "Person" з масиву елементів такого типу, в DOM HTML ми використовували оператор індексування. В DOM XML він також працює, але можна також скористатись методом **item(index)**. Наприклад:

```
var arr = document.getElementsByTagName("h1");

//доступ до вмісту першого тега <h1> засобами DOM HTML
document.write("header [0]: " + arr[0].innerText);

//доступ до вмісту першого елемента <h1> засобами DOM XML
document.write("header [0]: " + arr[0].innerText);
document.write("header [0]: " + arr.item(0).text);
```

Доречі, ви можете запустити JavaScript код для DOM в консолі. Для цього слід написати сценарій з використанням **Windows Script Host (WSH)**. WSH – це компонент ОС Windows, який призначений для запуску сценаріїв на скриптових мовах, таких як JScript, VBScript, Perl тощо. Вони являються заміною .bat файлам, та містять в основному скрипти для автоматизації певних дій адміністратора. Зокрема дозволяють працювати з файлами, реєстром, здійснювати запуск



програм, працювати з мережею та користувачами, наприклад, виводити список мережевих дисків, отримувати імена користувачів тощо. Але нас буде цікавити лише робота з вводом/виводом даних та з файловою системою.

Сам по собі WSH-сценарій - це **звичайний текстовий файл з розширенням *.js, *.vbs, *.wsf**. В якості мови написання сценарію оберемо JScript, який являється різновидом JavaScript. А тепер перейдемо власне до самого кодування.

Головним об'єктом WSH виступає **WScript**. Для введення та виведення даних він містить набір вбудованих методів та аргументів. Наприклад, виведення даних можна організувати за допомогою метода **echo**:

```
//синтаксис
object.echo ([Arg1] [,Arg2] [,Arg3] ...)

//приклад
WScript.echo("Hello, World");
```

Крім того, ввід та вивід інформації можна організувати за допомогою одного з наступних **об'єктів**:

- **stdin** – потік вводу;
- **stdout** – потік виводу;
- **stderr** – потік помилок.

Об'єкт **WScript.stdin** має методи **read()** і **readLine()**, які дозволяють зчитати один символ або цілий рядок вводу з клавіатури. Щодо останніх двох, то для виведення інформації вони використовують методи **write()** і **writeLine()**.

Для прикладу, перепишемо наш сценарій доступу до вузлів XML-документа.

testDOM.js

```
//створюємо екземпляр парсера MSXML
var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
//var doc = CreateObject("Microsoft.XMLDOM");

doc.async = false;
doc.validateOnParse = false;
doc.resolveExternals = false;

//завантажуємо XML документ для обробки
if(!doc.load("notebook.xml")) {
    WScript.echo("Error loading file!");
}

//виводимо текстовий зміст завантаженого XML документа
WScript.echo("DOMDocument (text): " + doc.text);

//доступаємось до елемента <person> та виводимо його текстовий зміст
WScript.echo("person: " + doc.getElementsByTagName("person").item(1).text);

var node = doc.getElementsByTagName("person").item(1);
WScript.echo("node name      : " + node.nodeName + "<br />");
WScript.echo("attribute id : " + node.getAttribute("id"));
```

Запустити даний сценарій на виконання можна за допомогою однієї з двох **утиліт**:

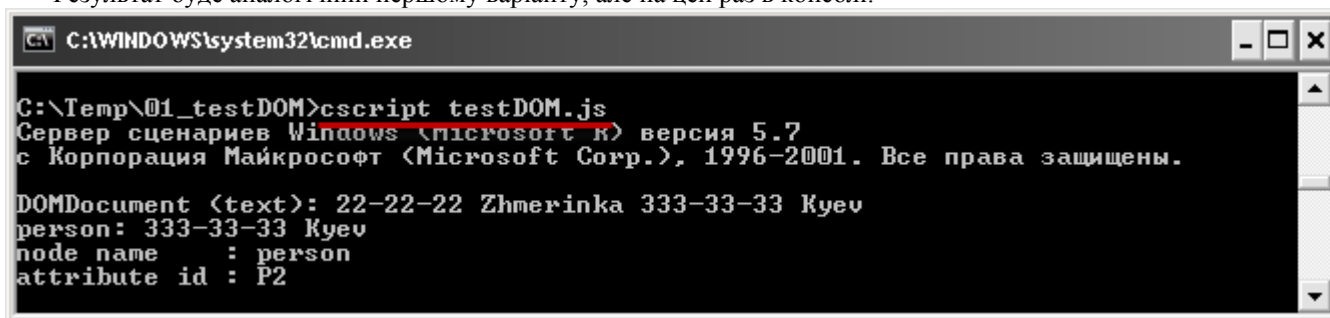
- 1) **wscript.exe (windows script code)**, яка має графічний інтерфейс для обробки сценарію, наприклад, дозволяє працювати з діалоговими вікнами.
- 2) **cscript.exe (console script code)**, яка має консольний інтерфейс для роботи з сценаріями.

Саме другою утилітою ми і скористаємось. Для цього слід в командному рядку виконати наступну команду:

```
cscript testDOM.js
```



Результат буде аналогічний першому варіанту, але на цей раз в консолі:



```

C:\Temp\01_testDOM>cscript testDOM.js
Сервер сценариев Windows (microsoft) версия 5.7
с Корпорация Майкрософт (Microsoft Corp.), 1996-2001. Все права защищены.

DOMDocument (text): 22-22-22 Zhmerinka 333-33-33 Kyev
person: 333-33-33 Kyev
node name      : person
attribute id   : P2
  
```

Всі приклади кодів можна буде переглянути в папці [sources\01_testDOM](#) представленого уроку.

4. Відловлення синтаксичних помилок

Ніхто не застрахований від помилок і для їх відловлення у парсера MSXML існує об'єкт **parseError** (він не являється частиною стандарту W3C DOM). Отримавши доступ до об'єкта `parseError`, можна отримати код помилки, текст, в якому вона міститься, і навіть номер рядка, в якому виникла помилка.

Об'єкт `parseError` має наступні основні **властивості**:

Властивість	Опис
<code>errorCode</code>	Код помилки типу long integer
<code>reason</code>	Опис помилки
<code>line</code>	Номер рядка, який викликав помилку (long integer)
<code>linePos</code>	
<code>srcText</code>	Рядок коду, який викликав помилку
<code>url</code>	URL адреса завантаженого документа
<code>filePos</code>	Позиція в файлі, де відбулась помилка (long integer)

Наприклад:

```

...
<body>
  <script type="text/javascript">
    <!--
    function main() {
      //завантажуємо XML документ для обробки
      if(!doc.load("notebook.xml"))
      {
        alert("Error loading file!");
        document.write("Error Code: " + doc.parseError.errorCode + "<br />");
        document.write("Error Reason: " + doc.parseError.reason + "<br />");
        document.write("Error Line: " + doc.parseError.line + "<br />");
        return false;
      }
      //виводимо текстовий вміст завантаженого XML документа
      document.write("<b>DOMDocument (text)</b>: " + doc.text + "<br /><br />");
      //доступаємось до елемента <person> та виводимо його текстовий вміст
      document.write("<b>person</b>: " +
        doc.getElementsByTagName("person").item(1).text);

      var node = doc.getElementsByTagName("person").item(1);
      document.write("<b>node name</b>      : " + node.nodeName + "<br />");
      document.write("<b>attribute id</b>: " + node.getAttribute("id"));

      return true;
    }

    main();
    <!-->
  </script>
</body>
</html>
  
```




5. Обхід дерева DOM

Для обходу дерева XML-документа, DOM використовує вже відомий вам набір **методів** та **властивостей**:

1. **object = document.getElementsByTagName ("назва_елем.")** - повертає вказівник, наприклад, з іменем **object** на елемент з визначеним іменем ("назва_елем.");
2. **object.parentNode** - повертає вказівник на батьківський елемент;
3. **object.previousSibling** - повертає вказівник на попередній елемент;
4. **object.nextSibling** - повертає вказівник на наступний елемент;
5. **object.hasChildNodes()** - метод, що дозволяє перевірити чи має даний вузол дочірні вузли. Повертає **true** - якщо він має дітей;
6. **object.childNodes** - масив, в якому зберігаються дочірні вузли;
7. звернутись до **першого** елемента можна одним з наступних способів:
 - **object.childNodes[0]**
 - **object.firstChild**
8. звернутись до **останнього** елемента можна наступним чином:
 - **object.childNodes[object.childNodes.length-1]**
 - **object.firstChild.nextSibling.nextSibling.....**
 - **object.lastChild**
9. звернутись до елемента **верхнього рівня**:
 - **object.parentNode.parentNode.....**
 - **document.documentElement**
10. **object.nodeName** - ім'я вузла-елемента або зарезервоване означення типу вузла (див. таблицку нижче);
11. **object.nodeValue** - містить значення вузла, якщо воно існує;
12. **object.nodeType** - тип вузла. Модель DOM визначає 12 типів вузлів. Приведемо їх список:

Об'єкт DOM	Опис	nodeName	nodeType	nodeValue
Node	Базовий тип для інших об'єктів	-	-	-
Element	Вузол елемента	Ім'я елемента	1	null
Attr або Attribute	Атрибут, який представлений парою «ім'я-значення»	Ім'я атрибута	2	Значення атрибута
Text	Текстова нода (вузол), який представляє текст, що міститься в елементі або атрибуті	#text	3	Текст батьківського компонента
CDATASection	Розділ CDATA	#cdata-section	4	Вміст розділу CDATA
Entity Reference	Посилання на сутність, яка має наступний вигляд: &Entity;	Ім'я для посилання на сутність	5	null
Entity	Сутність	Ім'я сутності	6	null
ProcessingInstruction	Інструкція по обробці	Призначення інструкції (target)	7	Вміст інструкції
Comment	Коментар	#comment	8	Текст коментаря
Document	Кореневий вузол документа (весь XML-документ)	#document	9	null
DocumentType	Оголошення типу документа (DTD)	Ім'я корневого елемента	10	null
Document Fragment	Фрагмент (частина) документу	#document-fragment	11	null
Notation	Нотація, яка міститься в DTD-документі або в схемі	Ім'я нотації	12	null

Отже, отримати ім'я кожного вузла можна за допомогою властивості **nodeName**. Імена, які починаються з (#), вказують на неіменовані компоненти XML-документа. Наприклад, коментар в XML-документі не має імені. В зв'язку з цим, DOM використовує стандартне ім'я **#comment**. У всіх інших випадках, імена вузлів відповідають іменам, які були їм присвоєні в XML-документі.

Отримати значення кожного вузла можна за допомогою властивості **nodeValue**, якщо він може мати значення.



Обхід дерева XML-документа та перегляд його вузлів – це найбільш часто зустрічаєма дія при роботі з документами. Давайте спробуємо звичними нам засобами здійснити рекурсивний обхід дерева вже знайомого XML-документа «Записна книга»:

notebook.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<notebook>
  <person id="P1">
    <fullname firstname="Vasja" surname="Pupkin"/>
    <phone>22-22-22</phone>
    <city type="смт">Zhmerinka</city>
  </person>
  <person id="P2">
    <fullname firstname="Dasha" surname="Ivanova"/>
    <phone>333-33-33</phone>
    <city type="м">Kyev</city>
  </person>
</notebook>
```

Сам код по перегляду ієрархічної структури XML документа буде виглядати наступним чином:

recurse.html

```
<html>
<head>
  <title>Test DOM XML</title>
  <script type="text/javascript">
    <!--
      //створюємо екземпляр парсера MSXML
      var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");

      doc.async = false;
      doc.validateOnParse = false;
      doc.resolveExternals = false;
    </script>
</head>
<body>
  <script type="text/javascript">
    <!--
      main();

      function main()
      {
        var line = prompt("Enter file name: ", "notebook.xml");
        if (!doc.load(line)){
          alert("Error loading file!");
          document.write("Error Code: " + doc.parseError.errorCode + "<br />");
          document.write("Error Reason: " + doc.parseError.reason + "<br />");
          document.write("Error Line: " + doc.parseError.line + "<br />");
        }
        walk(doc, 0);
      }

      function tab(n){
        for(var i=0; i<n; ++i)
          document.write("&nbsp;&nbsp;&nbsp;");
      }

      function walk(node, level)
      {
        tab(level);
        document.write("node: " + node.nodeName + "<br />");
```



```
if(node.nodeType == 1)
{
    //якщо нода - це елемент (тег)
    tab(level);
    document.write("element ");

    //перевіряємо наявність атрибутів та виводимо про них інформацію,
    //якщо вони наявні
    if(node.attributes.length == 0)
    {
        document.write("with no attributes <br />");
    } else {
        document.write("with " + node.attributes.length
                        + " attributes <br />");
        for(var i=0; i<node.attributes.length; ++i)
        {
            var attr = node.attributes.item(i);
            tab(level+1);
            document.write("attribute: " + attr.name + " = "
                            + attr.value + "<br />");
        }
    }
} else
if(node.nodeType == 3) //якщо нода - це текстовий вузол
{
    tab(level);
    document.write("text: " + node.text + "<br />");
} else
if(node.nodeType == 9) //якщо нода - це кореневий вузол док.
{
    document.write("document node <br />");
} else
if(node.nodeType == 7) //якщо нода-це інструкція по обробці
{
    tab(level);
    document.write("processing instruction: &lt;?"
                    + node.nodeName + " "
                    + node.nodeValue + ">"
                    + "<br />");
} else
if(node.nodeType == 8) //якщо нода - це коментар
{
    tab(level);
    document.write("comment: " + node.text + "<br/>");
}

//перевіряємо наявність дочірніх вузлів
if(node.hasChildNodes())
{
    for(var i=0; i<node.childNodes.length; ++i)
    {
        var child = node.childNodes.item(i);

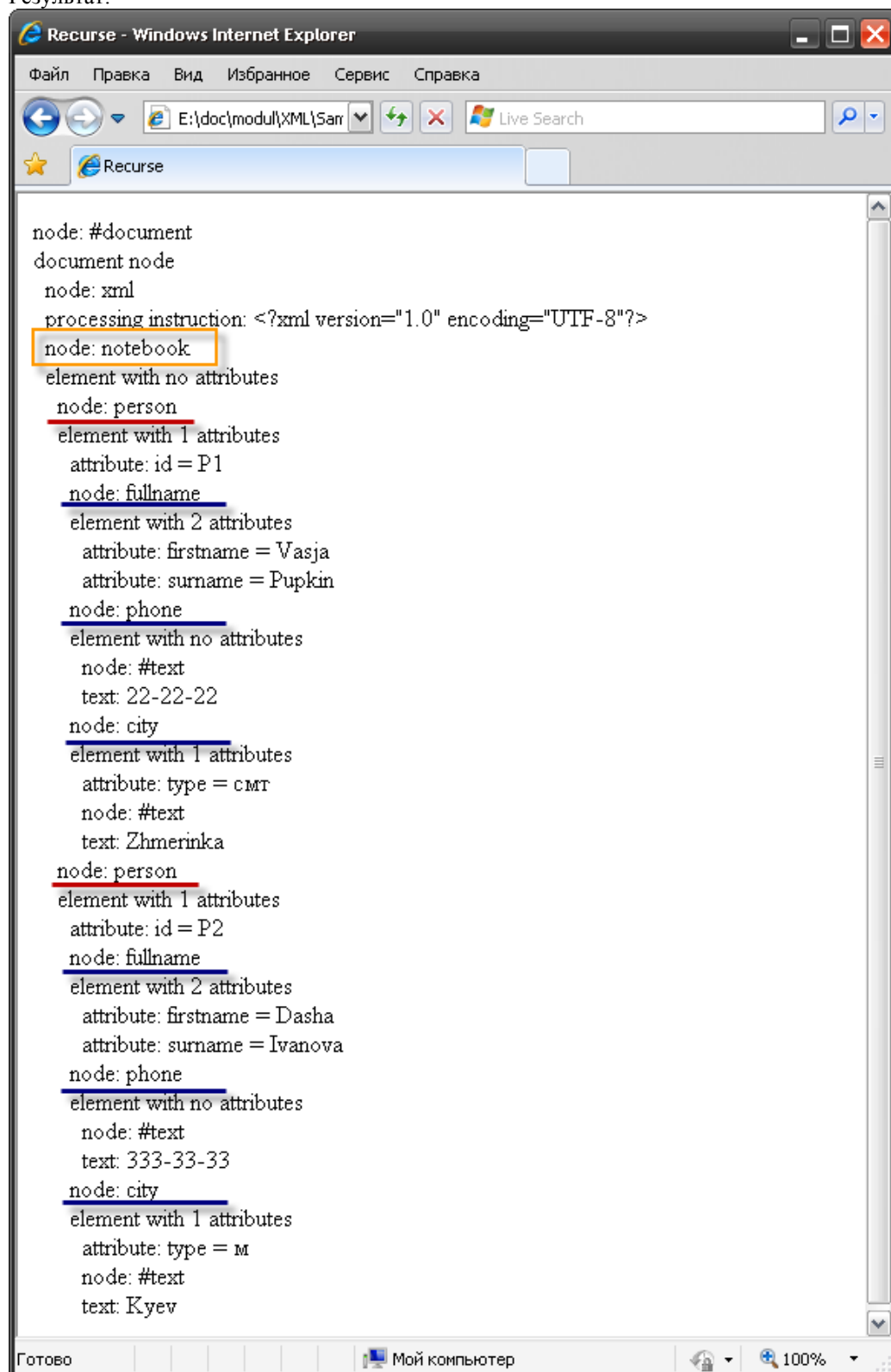
        //виводимо інформацію про кожен дочірній вузол
        walk(child, level+1);
    }
}

}

//-->
</script>
</body>
</html>
```



Результат:



```

node: #document
document node
  node: xml
    processing instruction: <?xml version="1.0" encoding="UTF-8"?>
    node: notebook
      element with no attributes
        node: person
          element with 1 attributes
            attribute: id = P1
            node: fullname
              element with 2 attributes
                attribute: firstname = Vasja
                attribute: surname = Pupkin
            node: phone
              element with no attributes
                node: #text
                  text: 22-22-22
            node: city
              element with 1 attributes
                attribute: type = cmr
                node: #text
                  text: Zhmerinka
          node: person
            element with 1 attributes
              attribute: id = P2
              node: fullname
                element with 2 attributes
                  attribute: firstname = Dasha
                  attribute: surname = Ivanova
              node: phone
                element with no attributes
                  node: #text
                    text: 333-33-33
              node: city
                element with 1 attributes
                  attribute: type = m
                  node: #text
                    text: Kyev

```

Всі файли коду можна переглянути в папці [sources\02_recurse](#).

6. Зміна структури XML документа

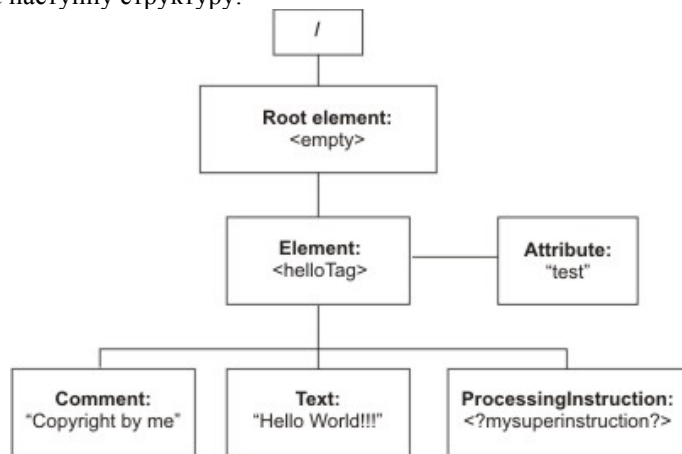
Крім вище перерахованих методів для обходу дерева XML документа, існує і ряд інших методів, які дозволяють здійснити різного роду маніпуляції з документом засобами DOM. Наступний представлений набір методів фактично дозволяє змінювати ієрархічну структуру документа. Розглянемо їх:

1. **document.createElement('nameElement')** – створює новий елемент (тег) в пам'яті (!). Значенням nameElement може бути ім'я довільного елемента: person, student, name тощо.
2. **document.createTextNode('text comment')** – створює текстовий коментар в пам'яті.



3. **object.setAttribute('nameAttribute', 'value')** – створює новий атрибут (з вказаним в другому параметрі значенням) для викликаючого елемента.
4. **document.appendChild(newElem)** – додає в XML-документ новий елемент (newElem). При цьому новий вузол буде доданий в кінець документа.
5. **parent.insertBefore('newElement', 'beforeElement')** - вставляє новий елемент (newElement) перед елементом beforeElement.
6. **parent.insertAfter('newElement', 'afterElement')** - вставляє новий елемент (newElement) після елемента afterElement.
7. **parent.removeChild(elemDelete)** – видаляє вказаний елемент (elemDelete) разом з всіма потомками.
8. **object_1.swapNode(object_2)** - міняє два елементи місцями (тільки для дочірніх елементів одного батька).
9. **object.cloneNode** - копіює вузол разом з дочірніми елементами та повертає посилання на новий елемент, який можна додати будь-яким іншим способом.
10. **object.removeNode()** - видаляє вузол з дерева, при цьому його потомки переміщуються в батьківський для видаленого вузол.

Отже, напишемо скрипт, який дозволить створити динамічно XML-документ. Результатом буде XML документ, який має наступну структуру:



create.html

```
<html><head>
  <title>Test DOM XML</title>
  <script type="text/javascript">
    <!--
    function CreateXML()
    {
      var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
      doc.async = false;
      doc.validateOnParse = false;
      doc.resolveExternals = false;

      //завантажуємо XML документ
      //doc.load("empty.xml");
      doc.loadXML("<?xml version='1.0'?><empty></empty>");

      var tag = doc.createElement("helloTag"); //створюємо елемент з іменем
                                              //helloTag
      tag.setAttribute("test", "failed"); //створюємо йому атрибут test = failed
      var text = doc.createTextNode("Hello World!!!"); //створюємо текстову ноду
      tag.appendChild(text); //встановлюємо текстову ноду в елемент helloTag

      var comment = doc.createComment("COPYRIGHT BY ME"); //створюємо коментар
      tag.insertBefore(comment, text); //вставляємо коментар перед текстовою нодою

      //створюємо та встановлюємо інструкцію по обробці в елемент helloTag
      var pi = doc.createProcessingInstruction("mysuperinstruction",
                                              " prop='value' ");
      tag.appendChild(pi);
    }
  </script>
</head></html>
```



```

        doc.documentElement.appendChild(tag);    //додаємо на сторінку кореневий
                                                //елемент helloTag

        //записуємо створений xml документ з іменем nonempty.xml
        doc.save("noempty.xml");
    }
    //-->
</script>
</head>
<body>
    <button onclick="CreateXML();">Create new XML</button>
</body>
</html>

```

Результатом буде створення нового XML-документа **noempty.xml**:

```

<?xml version='1.0' ?>
<empty>
    <helloTag test="failed">
        <!--COPYRIGHT BY ME-->
        Hello World!!!
        <?mysuperinstruction prop='value' ?>
    </helloTag>
</empty>

```

Коди сценаріїв можна отримати в папці [sources\03_create](#).

7. Відбір вузлів за допомогою XPath. XSLT трансформація засобами DOM

Парсер MSXML містить також набір методів, які дозволяють здійснити відбір певного набору вузлів. Для пошуку необхідного набору вузлів використовуються XPath вирази, а також **2 методи** XML DOM:

- **selectNodes(XPath-вираз)** – повертає колекцію типу XMLDOMNodeList, яка містить колекцію елементів, обраних вказаним XPath виразом;
- **selectSingleNode(XPath-вираз)** – повертає перший вузол з обраної колекції елементів, які відповідають вказаному XPath виразу.

Розглянемо невеличкий приклад пошуку вузлів в нижчепредставленому XML-документі:
users.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<users>
    <user id="U-01">
        <name>Том Джонс</name>
        <login>tom</login>
        <password>Zder12</password>
        <role>user</role>
    </user>
    <user id="U-02">
        <name>Іван Сусанін</name>
        <login>suslik</login>
        <password>qwerty</password>
        <role>admin</role>
    </user>
    <user id="U-03">
        <name>Том Робінс</name>
        <login>robin</login>
        <password>123</password>
        <role>user</role>
    </user>
    ...
</users>

```




xpath.html

```

<html><head>
  <title>XPath samples</title>
  <script type="text/javascript">
    <!--
    function Search()
    {
      var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
      doc.async = false;
      doc.validateOnParse = false;
      doc.resolveExternals = false;

      //завантажуємо XML документ
      var filename = prompt("Введіть назву XML документа: ", "users.xml");
      doc.load(filename);

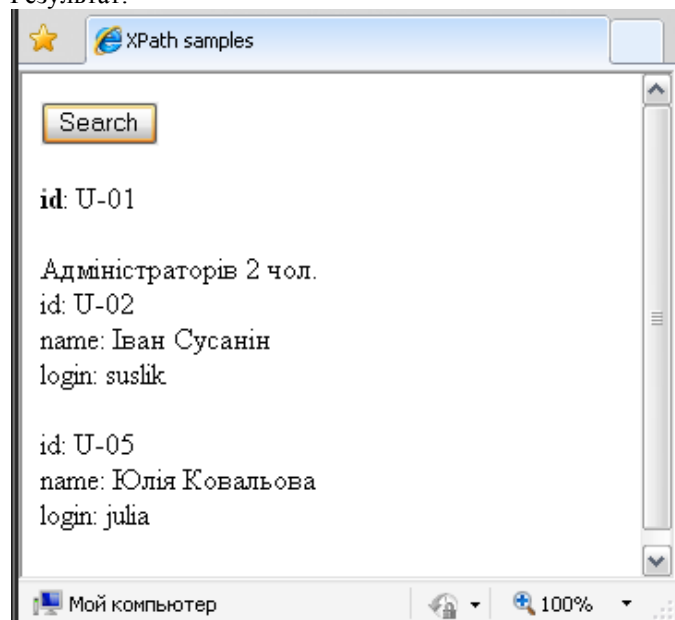
      //виводимо ідентифікатор другого користувача
      var attr = doc.selectSingleNode("/users/user[1]/@id");
      var para = document.getElementById("para");
      para.innerHTML = "<b>id</b>: " + attr.value + "<br />";

      //відбираємо всіх адміністраторів
      //та виводимо їх імена і логіни
      var nodes = doc.selectNodes("/users/user[role = 'admin']");
      para.innerHTML += "<br />Адміністраторів " + nodes.length + " чол.<br />";

      for(var i=0; i<nodes.length; i++) {
        var user = nodes.item(i);
        para.innerHTML += "id: " + user.getAttribute("id") + "<br />";
        //можна так
        para.innerHTML += "name: " + user.childNodes[0].text + "<br />";
        //а можна і так
        para.innerHTML += "login: " + user.selectSingleNode("login").text;
        para.innerHTML += "<br /><br />";
      }
    }
    <!-->
  </script>
</head>
<body>
  <button onclick="Search();">Search</button>
  <p id="para"></p>
</body>
</html>

```

Результат:





Скриптовый файл для аналогічних цілей на WSH переписється так:

xpath.js

```
function Search()
{
    var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
    doc.async = false;
    doc.validateOnParse = false;
    doc.resolveExternals = false;

    //завантажуємо XML документ
    doc.load("users.xml");

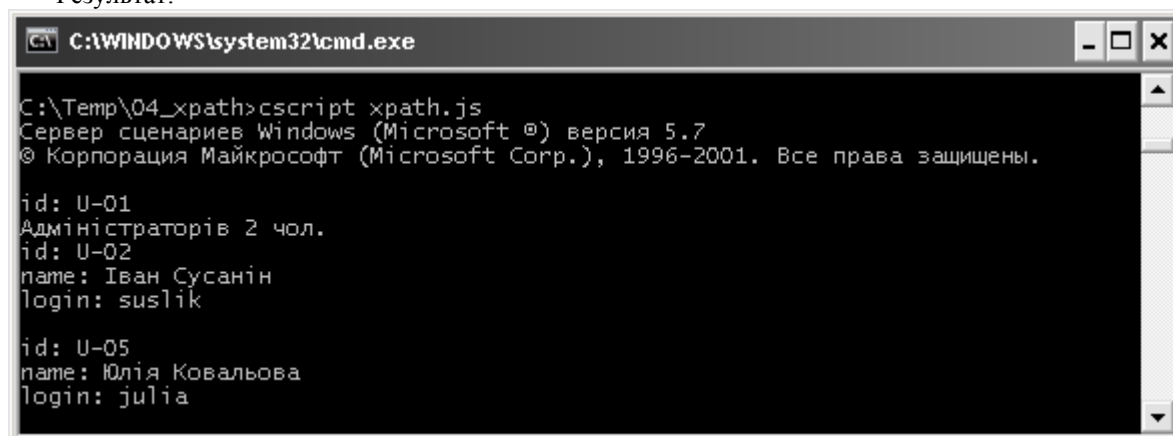
    //виводимо ідентифікатор другого користувача
    var attr = doc.selectSingleNode("/users/user[1]/@id");
    WScript.echo("id: " + attr.value);

    //відбираємо всіх адміністраторів
    //та виводимо їх імена і логіни
    var nodes = doc.selectNodes("/users/user[role = 'admin']");
    WScript.echo("Адміністраторів " + nodes.length + " чол.");

    for(var i=0; i<nodes.length; ++i) {
        var user = nodes.item(i);
        WScript.echo("id: " + user.getAttribute("id"));
        WScript.echo("name: " + user.childNodes[0].text);
        WScript.echo("login: " + user.selectSingleNode("login").text);
        WScript.echo();
    }
}

Search();
```

Результат:



```
C:\WINDOWS\system32\cmd.exe
C:\Temp\04_xpath>cscript xpath.js
Сервер сценариев Windows (Microsoft ©) версия 5.7
© Корпорация Майкрософт (Microsoft Corp.), 1996-2001. Все права защищены.

id: U-01
Адміністраторів 2 чол.
id: U-02
name: Іван Сусанін
login: suslik

id: U-05
name: Юлія Ковальова
login: julia
```

MSXML містить також набір методів для обробки XML-документів згідно створеної таблиці стилів. До таких **методів відносять**:

- **transformNode(stylesheet)** – застосовує вказану таблицю стилів до завантаженого XML документа. Метод повертає результат обробки у вигляді рядка.
- **transformNodeToObject(stylesheet, outputObject)** – аналогічний попередньому методу, але результат обробки повертається в другий параметр outputObject. В основному використовується для трансформації XML-XHTML.

Для того, щоб протестувати дані методи, для початку напишемо таблицю стилів до XML-документа з інформацією про користувачів.

users.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" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
```



```

<html>
  <head>
    <title>Users</title>
  </head>
  <body>
    <xsl:apply-templates select="/users" />
  </body>
</html>
</xsl:template>

<xsl:template match="users">
  <table border="1">
    <tbody>
      <tr style="background-color: lightgray;">
        <th>Код</th>
        <th>Повне ім'я</th>
        <th>Логін</th>
        <th>Пароль</th>
      </tr>
      <xsl:apply-templates select="/users/*" />
    </tbody>
  </table>
</xsl:template>

<xsl:template match="user">
  <tr>
    <xsl:choose>
      <xsl:when test="role = 'user'">
        <xsl:attribute name="style">
          background-color: white
        </xsl:attribute>
      </xsl:when>
      <xsl:when test="role = 'student'">
        <xsl:attribute name="style">
          background-color: lightblue
        </xsl:attribute>
      </xsl:when>
      <xsl:when test="role = 'admin'">
        <xsl:attribute name="style">
          background-color: red
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="style">
          background-color: magenta
        </xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>

    <td><xsl:value-of select="@id" /></td>
    <td><xsl:value-of select="name" /></td>
    <td><xsl:value-of select="login" /></td>
    <td><xsl:value-of select="password" /></td>
  </tr>
</xsl:template>
</xsl:stylesheet>

```

А тепер використаємо нашу таблицю стилів для відображення користувачів на Web-сторінці у табличному вигляді. Нехай новоутворена HTML сторінка буде зберігатись на диску D:\. В такому разі необхідний для конвертації код буде мати нижчеописаний вигляд:



xslt.html

```
<html>
<head>
  <title>Test DOM XML</title>
  <script type="text/javascript">
    <!--
      ...
      //-----
      function createDom(fileName) {
        var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
        doc.async = false;
        doc.validateOnParse = false;
        doc.resolveExternals = false;

        if (fileName != null)
          doc.load(fileName);
        return doc;
      }
      function Transform() {
        //створюємо посилання на XML-документ та таблицю стилів
        var filename = prompt("Введіть шлях до XML-документа: ", "users.xml");
        var doc = createDom(filename);
        filename = prompt("Введіть шлях до XSLT-документа: ", "users.xslt");
        var xsl = createDom(filename);

        //трансформуємо XML-документ згідно завантаженої таблиці стилів
        var result = doc.transformNode(xsl);

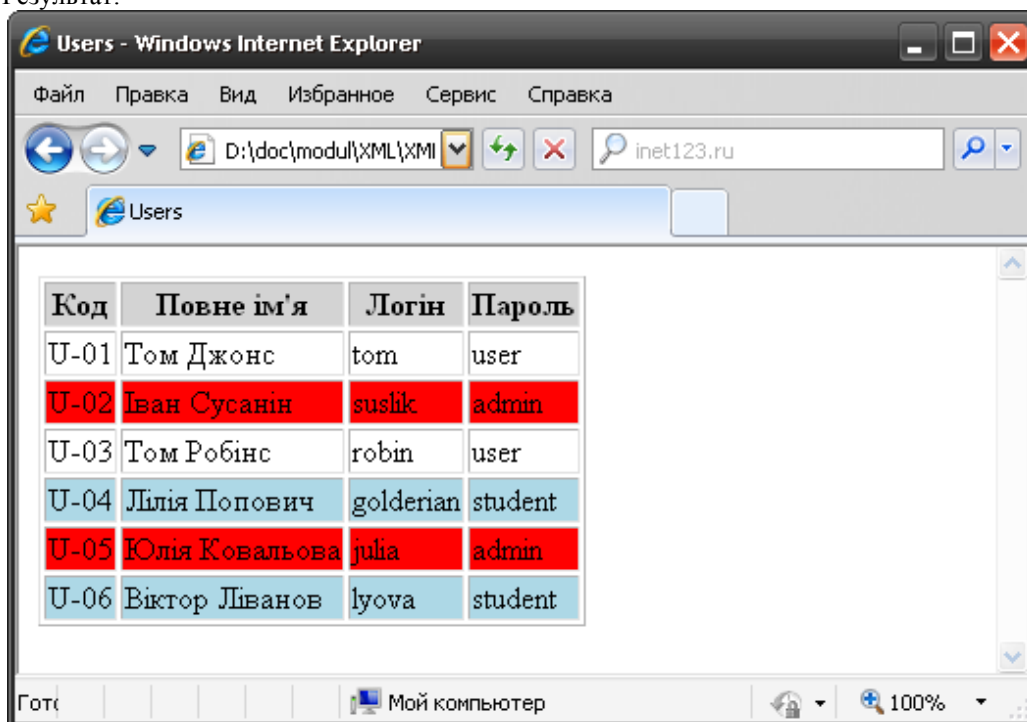
        //створюємо об'єкт FSO для роботи з файловою системою
        var fso = new ActiveXObject("Scripting.FileSystemObject");

        //створюємо файл або отримуємо посилання на існуючий
        var file = fso.createTextFile("D:\\result.html", true);
        //var file = fso.openTextFile("D:\\result.html", 2, true);
        file.write(result); //записуємо результат трансформації
        file.close();      //закриваємо файл

        var para = document.getElementById("para");
        para.innerText = "XML документ успішно конвертований";
      }
    <!-->
  </script>
</head>
<body>
  <button onclick="Search();">Search</button><br />
  <button onclick="Transform();">Transformation</button><br />
  <p id="para"></p>
</body>
</html>
```



Результат:



В нашому коді для збереження файлу на диск ми скористались об'єктом WSH для роботи з файловою системою **File System Object (Scripting.FileSystemObject)**. Цей об'єкт містить набір методів для створення директорій, файлів, їх копіювання, видалення тощо. Наприклад, щоб створити файл або директорію необхідно використати методи **CreateTextFile()** або **CreteFolder()** відповідно:

```
object.CreateFolder(foldername);
//object      - ім'я об'єкта FileSystemObject.
//foldername  - ім'я папки, яку потрібно створити

object.CreateTextFile(filename[, overwrite[, unicode]]);
//object      - ім'я об'єкта FileSystemObject або посилання на директорію
//filename    - ім'я створюваного файлу
//overwrite   - логічне значення, яке визначає чи перезаписувати файл
//            у випадку його існування
//unicode     - логічне значення, що вказує на підтримку Unicode
```

Якщо необхідно створити об'єкти для вже існуючої папки або файла, тобто отримати посилання на них, тоді можна скористатись методами **GetFile()** та **GetFolder()**.

```
object.GetFile( path );
object.GetFolder( path );
```

Де path – шлях до вже існуючої директорії чи файла.

Наприклад:

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var file = fso.GetFile("C:\\myfile.bat");
```

Після того, як файл або директорію створили чи отримали на нього посилання можна переглядати їх вміст та редагувати. Оскільки файли являються об'єктами файлової системи, перед використанням методів для роботи з ними, необхідно спочатку проініціалізувати об'єкт **FileSystemObject**. Наприклад, метод **OpenTextFile()** відкриває файл, але якщо його не існує, тоді він створюється. З параметрів, які передаються в даний метод обов'язковим є лише перший – ім'я файла.

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
//object      - ім'я об'єкта FileSystemObject
//filename    - ім'я файла для відкриття
//iomode      - режим відкриття, наприклад, ForReading(1), ForWriting(2) або ForAppending(8)
//create      - логічне значення; створювати новий файл чи ні
//format      - кодування для відкриття файлу: системне (2), Unicode (1), ASCII (0)
```



Для запису даних у відкритий файл використовуються методи:

- ✓ **Write** – записує дані в файл в один рядок, без переведу курсора;
- ✓ **WriteLine** – записує в файл окремий рядок;
- ✓ **WriteBlankLines** – записує пустий рядок. В якості параметра приймає кількість пустих рядків, які потрібно записати.

Для зчитування відкритого файла використовують методи:

- **Read** – читає з файла вказану в якості параметра кількість символів;
- **ReadLine** – зчитує рядок з файла;
- **ReadAll** – зчитує весь вміст файла;
- **Skip** – пропускає вказану кількість символів;
- **SkipLine** – пропускає рядок.

Наприклад:

```
var FSO = WScript.CreateObject("Scripting.FileSystemObject"); //створюємо об'єкт FSO

var file = FSO.OpenTextFile("C:\\test.txt", 2, true); //відкриваємо файл для запису.
                                                    //Якщо файла не існує, створюємо його
file.Write("Hello, World!"); //записуємо текст в файл
file.WriteBlankLines(2); //записуємо 2 пустих рядки
file.WriteLine("La-la-la. The End! ;-)"); //записує рядок даних в файл
file.Close(); //закриваємо файл
```

А тепер спробуємо переписати код застосування таблиці стилів до XML-документа на WSH:

xslt.js

```
var cin = WScript.stdIn;
var cout = WScript.stdOut;

function createDom(fileName) {
    var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
    doc.async = false;
    doc.validateOnParse = false;
    doc.resolveExternals = false;

    if (fileName != null)
        doc.load(fileName);
    return doc;
}

function main() {
    //створюємо посилання на XML-документ та таблицю стилів
    cout.write("Введіть шлях до XML-документа: ");
    var filename = cin.readLine();
    var doc = createDom(filename);

    cout.write("Введіть шлях до XSLT-документа: ");
    filename = cin.readLine();
    var xsl = createDom(filename);

    //трансформуємо XML-документ згідно завантаженої таблиці стилів
    var result = doc.transformNode(xsl);

    //створюємо об'єкт FSO для роботи з файловою системою
    var fso = new ActiveXObject("Scripting.FileSystemObject");
    //створюємо файл або отримуємо посилання на існуючий
    var file = fso.createTextFile("D:\\result.html", true);
    //var file = fso.openTextFile("D:\\result.html", 2, true);
    file.write(result); //записуємо результат трансформації
    file.close(); //закриваємо файл

    cout.writeLine(result);
}

main();
```




Метод **transformNodeToObject(styleshet, outputObject)** також дозволяє здійснити XSLT-трансформацію XML-документа. Але результат трансформації повертається в другий параметр даного метода – outputObject і має вигляд об'єкта DOMDocument. Отже, даний метод може використовуватись лише для трансформації з XML в XML.

Особливої відмінності у використанні методів transformNodeToObject та transformNode немає. Приведемо невеличкий приклад того як зміниться код функції main() у випадку використання трансформації з XML в XML:

```
function main() {
    //створюємо посилання на XML-документ та таблицю стилів
    cout.write("Введіть шлях до XML-документа: ");
    var filename = cin.readLine();
    var doc = createDom(filename);

    cout.write("Введіть шлях до XSLT-документа: ");
    filename = cin.readLine();
    var xsl = createDom(filename);

    //трансформуємо XML-документ згідно завантаженої таблиці стилів
    var result = new ActiveXObject("MSXML2.DOMDocument.6.0");
    result.async = false;
    result.validateOnParse = false;
    result.resolveExternals = false;

    doc.transformNodeToObject(xsl, result);

    //записуємо результат трансформації в файл
    result.save("result.xml");

    cout.writeLine(result.text);
}
```

Ось і все.

8. Робота з DOM XML в інших браузерах

Але, у всьому потрібно бути універсальним, навіть у написанні скриптів для перегляду та роботи з XML-документами. Адже, якщо вже щось робите, то потрібно, щоб воно працювало у всіх браузерах, а не лише в Internet Explorer.

Отже, спробуємо розібрати як зробити код для DOM XML універсальним. Для цього консорціум W3C рекомендує робити перевірку на підтримку браузерами певних об'єктів: для IE перевірка на підтримку ActiveX-об'єктів, а для всіх інших – перевірка на підтримку об'єкта implementation та його метода createDocument().

```
if (window.ActiveXObject){
    //код для Internet Explorer
}else
    if (document.implementation && document.implementation.createDocument){
        //код для Mozilla, Firefox, Opera тощо
    }
```

Однак, така перевірка не зовсім коректна, оскільки для браузерів Chrome та Safari, тобто для браузерів [WebKit](#), він не буде коректно працювати. В зв'язку з цим для завантаження та роботи з XML-документом, пропонуємо скористатись об'єктом [XMLHttpRequest](#), який підтримується всіма браузерами. Однак методи його будуть в деяких випадках відрізнятись для браузера Internet Explorer, тому невеличку перевірку все ж необхідно буде робити.

Потрібно також врахувати, що об'єкт XMLHttpRequest працює на клієнті і для його роботи потрібен веб-сервер.

Отже, для початку спробуємо написати універсальний код для завантаження XML-документів та пошуку необхідних даних, включаючи використання XPath виразів. Щоб відчуті різницю, XML-документ візьмемо той самий, тобто **users.xml**.

Оскільки об'єкт XMLHttpRequest в основному використовується для AJAX, принцип його створення буде схожий з стандартним шаблоном AJAX додатку.

index.html

```
<html>
<head>
    <title>XPath samples</title>
    <script type="text/javascript">
    <!--
```



```

var xmlHttp = 0;
function createXMLHttpRequest()
{
    //створюємо об'єкт XmlHttpRequest
    try { //для IE 5, 6
        xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
    } catch (e) {
        try { //для IE 7
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) { //для Firefox, Opera, Safari, Chrome тощо
            xmlHttp = new XMLHttpRequest();
        }
    }
    //або так
    //if(window.XMLHttpRequest)
    //    xmlHttp = new XMLHttpRequest();
    //else if(window.ActiveXObject)
    //    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

    //перевіряємо чи об'єкт створився
    if (!xmlHttp) {
        alert("Помилка при створенні XMLHTTP");
    }
}

function handleSearchXPath()
{
    //після того як дані завантажені: readyState = 4
    //і статус завантаження "OK": status = 200
    if(xmlHttp.readyState == 4 && xmlHttp.status == 200)
    {
        document.getElementById("para").innerHTML = "Complete.<br />";
        doc = xmlHttp.responseXML;
        var user = doc.documentElement.getElementsByTagName("user");
        var para = document.getElementById("para");
        var textUser = "";

        para.innerHTML += "В XML документах "+user.length+" користувачів<br/>";
        for(var i=0; i<user.length; i++)
        {
            //перевірку обов'язково спочатку робити на IE,
            //оскільки з версії IE8 підтримує XMLHttpRequest
            if(window.ActiveXObject) //для IE
                textUser += i + ": " + user[i].childNodes.item(0).text
                    + "<br />";
            else if(window.XMLHttpRequest) //для Firefox, Opera, Chrome тощо
                textUser += i + ": "
                    + user[i].getElementsByTagName("name")[0].textContent
                    + "<br />";
        }
        para.innerHTML += textUser;

        //-----
        //виводимо ідентифікатор другого користувача та логіни всіх студентів.
        if(window.ActiveXObject)
        {
            var result = doc.selectSingleNode("/users/user[1]/@id");
            para.innerHTML += "<br /><b>users/user[1]/@id</b>: "
                + result.value + "<br />";

            result = doc.selectNodes("/users/user[role='student']/login");
            for(var i=0; i<result.length; i++)
                para.innerHTML += "<br /><b>login</b>: " + result[i].text;
        }
    }
}

```



```

    }else
    {
        if(window.XMLHttpRequest)
        {
            var result = doc.evaluate("/users/user[1]/@id", doc, null,
                                    XPathResult.ANY_TYPE, null);
            var obj = result.iterateNext();
            para.innerHTML += "<br /><b>users/user[1]/@id</b>: " +
                            obj.textContent + "<br />";

            result = doc.evaluate("/users/user[role='student']/login", doc,
                                null, XPathResult.ANY_TYPE, null);
            obj = result.iterateNext();
            while(obj){
                para.innerHTML += "<br /><b>login</b>: "+ obj.textContent;
                obj = result.iterateNext();
            }
        }
    }else
    {
        document.getElementById("para").innerText
            = "Не вдалось отримати дані " + xmlhttp.status;
    }

function SearchXPath()
{
    //завантажуємо XML документ
    var filename = prompt("Введіть назву XML документа: ", "users.xml");

    createXMLHttpRequest();
    //вказуємо функцію-обробник відповіді
    xmlhttp.onreadystatechange = handleSearchXPath;
    xmlhttp.open("get", filename, true /*асинхронний чи синхронний*/);
    xmlhttp.send(null);
}
//-->
</script>
</head>
<body>
    <button onclick="SearchXPath();">XPath demo</button><br />
    <p id="para"></p>
</body>
</html>

```

Щоб побачити результат для запуску HTML-сторінки потрібно буде прописати шлях, який матиме один з представлених варіантів вигляду:

```

http://localhost/віртуальна_папка/імя_файла.html
http://ІПадреса_сервера/віртуальна_папка/імя_файла.html
http://імя_сервера/віртуальна_папка/імя_файла.html

```

Наприклад:

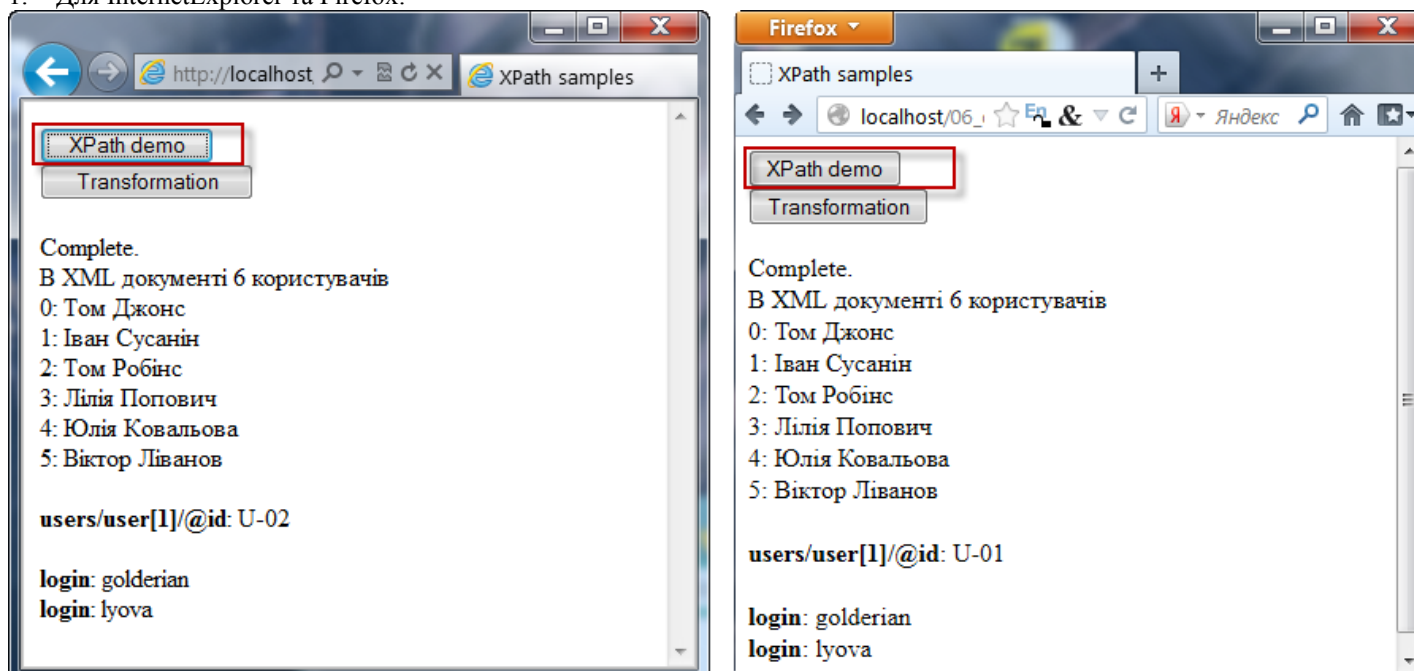
```
http://localhost/06_crossbrowser/index.html
```

При цьому папка 06_crossbrowser повинна лежати в директорії по замовчуванню, визначеною для використовуваного веб-сервера (читай [веб-сервер](#)).

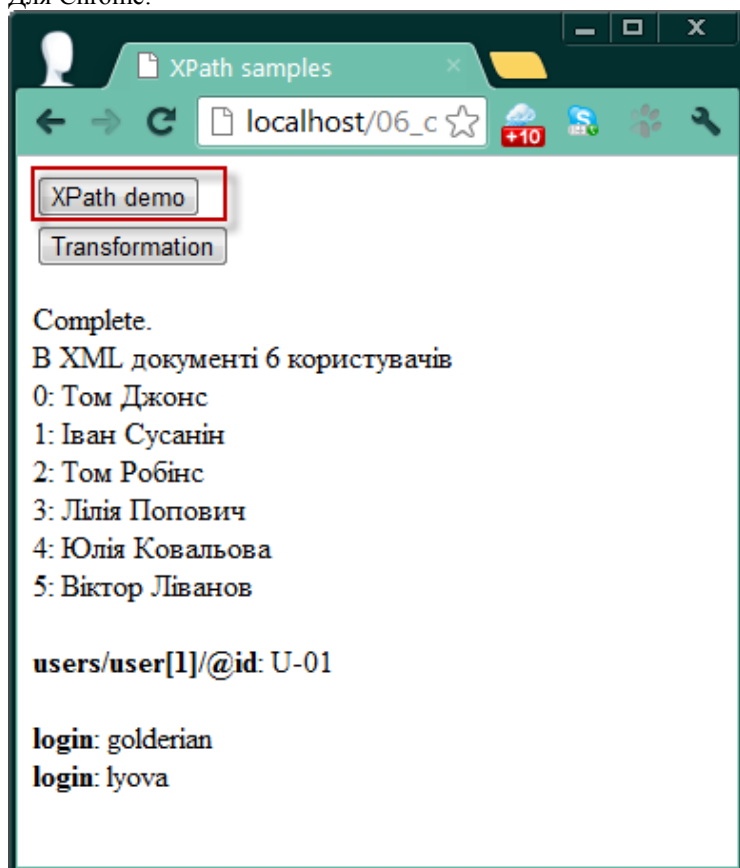


Результат буде таким:

1. Для InternetExplorer та Firefox:



2. Для Chrome:



Як бачите, даний код буде працювати однаково в браузерях Internet Explorer, Opera, Firefox, Chrome тощо.

А тепер спробуємо здійснити XSLT-трансформацію XML-документа з списком користувачів згідно розробленої таблиці стилів **users.xslt**. Для цього необхідно завантажити обидва документа з сервера і потім за допомогою методів об'єкта **XSLTProcessor** здійснити трансформацію. В такому разі модифікуємо метод `createXMLHttpRequest()`, який буде повертати посилання на створений об'єкт `XmlHttpRequest` для кожного документа. Напишемо також метод `loadXML()` для спрощення процесу створення об'єктів на XML- та XSLT-документ.

Для збереження результату можна скористатись класом **XMLSerializer**. Для прикладу, результат виведемо на сторінку.



index.html

```
<html><head>
  <title>XPath samples</title>
  <script type="text/javascript">
    <!--
    function createXMLHttpRequest()
    {
      var xmlHttp = 0;
      //створюємо об'єкт XmlHttpRequest
      try { //для IE 5, 6
        xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
      } catch (e) {
        try { //для IE 7
          xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) { //для Firefox, Opera, Safari, Chrome тощо
          xmlHttp = new XMLHttpRequest();
        }
      }
      //...
      //перевіряємо чи об'єкт створився
      if (!xmlHttp) {
        alert("Помилка при створенні XMLHTTP");
        return false;
      }
      return xmlHttp;
    }
    ...
    //-----
    function loadXML(filename)
    {
      var xmlHttp = createXMLHttpRequest();
      xmlHttp.open("get", filename, false);
      xmlHttp.send(null);

      return xmlHttp.responseXML;
    }

    function Transform()
    {
      //створюємо посилання на XML-документ та таблицю стилів
      var filename = prompt("Введіть шлях до XML-документа: ", "users.xml");
      var xmlContent = loadXML(filename);
      filename = prompt("Введіть шлях до XSLT-документа: ", "users.xslt");
      var xslContent = loadXML(filename);

      //трансформуємо XML-документ згідно завантаженої таблиці стилів
      var result = "";
      if (window.ActiveXObject) { //для IE
        result = xmlContent.transformNode(xslContent);
      } else if (window.XSLTProcessor) { //для Firefox, Opera, Safari тощо
        var xsltProcessor = new XSLTProcessor();
        xsltProcessor.importStylesheet(xslContent);

        var resultDocument = xsltProcessor.transformToDocument(xmlContent);
        var xmls = new XMLSerializer();
        result = xmls.serializeToString(resultDocument);
      }
      var para = document.getElementById("para");
      para.innerHTML += "XML документ успішно конвертований<br />";
      para.innerHTML += result;
    }
    <!-->
  </script>
```

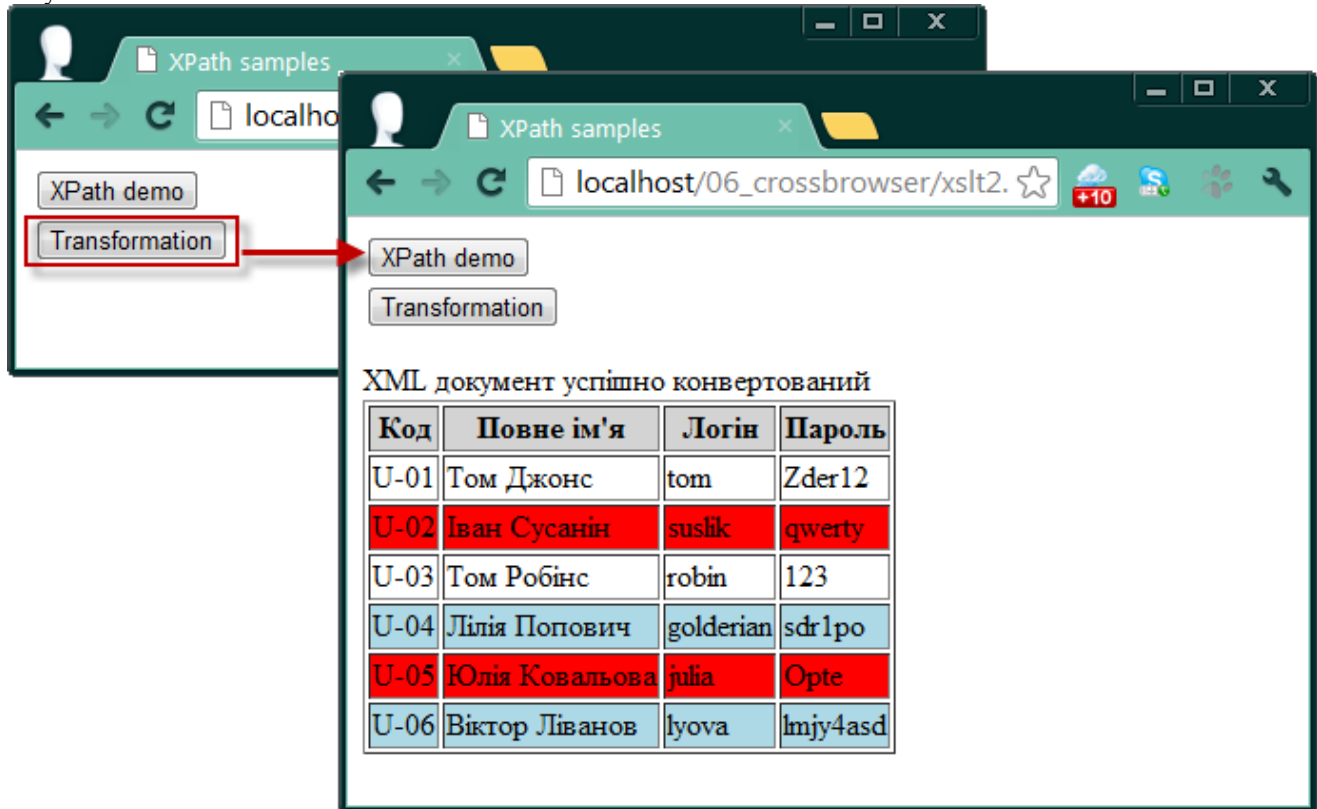


```

</head>
<body>
  <button onclick="SearchXPath();">XPath demo</button><br />
  <button onclick="Transform();">Transformation</button><br />
  <p id="para"></p>
</body>
</html>

```

Результат:



9. Домашнє завдання

По [шаблону XML документа](#), який описує звичайний клас C/C++, написати свій власний клас. Написаний документ засобами DOM необхідно трансформувати в звичайний файл реалізації (.cpp) з кодом.