**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ**

# *МЕТОДИЧНІ ВКАЗІВКИ*

***до виконання лабораторних робіт***

***з дисципліни XML- ТЕХНОЛОГІЇ***

для студентів IV курсу

Рівень вищої освіти – «Бакалавр»

Спеціальність – 122 – «Комп'ютерні науки»

ЗАТВЕРДЖЕНО

на засіданні групи забезпечення
спеціальності 122 Комп'ютерні науки
від « ___ » _____ 2020 року
протокол № ___
Голова групи_____(Мещеряков В.І.)

Затверджено на засіданні
кафедри Інформатики
Протокол №___від_____
Зав. кафедри _____Мещеряков В.І.

Одеса, 2020

***МЕТОДИЧНІ ВКАЗІВКИ***

***до виконання лабораторних робіт***

***з дисципліни XML- ТЕХНОЛОГІЇ***

для студентів IV курсу

Рівень вищої освіти – «Бакалавр»

Спеціальність – 122 – «Комп'ютерні науки»

Одеса, 2020

Методичні вказівки до виконання лабораторних робіт з дисципліни «***XML-технології***» для студентів 4-го курсу рівня вищої освіти «Бакалавр», за спеціальністю 122 – «Комп'ютерні науки»

Укладачі:

Гнатовська Г.А., к.т.н., доцент кафедри інформатики

**ЗМІСТ**

# ПЕРЕДМОВА

Дисципліна “XML-технології” викладається для рівня вищої освіти – «Бакалавр» спеціальності 122 – «Комп’ютерні науки» входить до складу вибіркової частини навчального плану підготовки бакалаврів. Викладається відповідно до робочого навчального плану підготовки бакалаврів.

Мета методичних вказівок – забезпечити отримання студентами теоретичних знань і практичних навичок щодо використання новітніх інформаційні технологій (IT) для вирішення практичних завдань структурування, передачі і подання інформації на базі сучасних XML-технологій. Набуття навичок впровадження XML-стандартів в реальні сучасні інформаційні, комунікаційні, бізнес рішення, втілюючи всі переваги компонентних технологій.

В процесі навчання основна увага направлена на використання XML, насамперед, як засобу для інтеграції існуючих стандартів зберігання і представлення даних і спрощення процесів обміну інформацією.

Ці методичні вказівки містять теоретичні відомості та розглянуті приклади виконання лабораторних робіт з дисципліни «XML-технології» та завдання. В методичних вказівках розглядаються питання, які відповідають навчальній програмі дисципліни.

# 1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО ДИСЦИПЛІНУ

## *1.1 Мета дисципліни та її місце у навчальному процесі*

Сучасний стан розвитку інформаційні технології вимагає знання та вміння застосовувати засоби забезпечення інтеграції інформаційних систем, що забезпечуються використанням XML-технологій, як стандартів для побудови складних структур даних у територіально-розподілених інформаційних системах з величезними обсягами інформації.

**Метою дисципліни** є ознайомлення студентів з основами структурування, передачі і подання інформації на базі сучасних XML-технологій та впровадження XML-стандартів в інформаційні, комунікаційні, бізнес рішення, втілюючи всі переваги компонентних технологій.

В процесі навчання основна увага направлена на використання XML насамперед як засобу для інтеграції існуючих стандартів зберігання і представлення даних і спрощення процесів обміну інформацією. В результаті вивчення дисципліни “ XML- технології ” студенти повинні:

ЗНАТИ:

- платформу XML та ii стандарти
- принципи розробки XML-документів;
- технології представлення та обробки даних в форматі XML;
- мову визначення схем XSD та DTD схем
- мову завдання шляхів XPath

ВМІТИ:

- оперувати основними поняттями сімейства стандартів та технологій XML
- застосовувати можливості XML-платформи та стандартів XML
- застосовувати мову XML для створення XML-документів
- застосовувати основні правила і конструкції для створення XML-схем та DTD-схем
- конструкції XSL для відображенняи XML-документів;
- застосовувати мову завдання шляхів XPath;
- верифікувати структури XML-документів

***Володіти компетенціями:*** знання XML-технологій і стандартів на основі мови XML для побудови складних структур даних у територіально-розподілених інформаційних системах з величезними обсягами інформації та у веб-сервісах.

*По кожній лабораторній роботі студент повинен скласти звіт, який містить в собі:*

- Назву роботи. Мету.

- Умову завдання згідно варіанту.
- Хід виконання роботи.
- Відповіді на контрольні питання.

*Оформлений звіт захищається студентом усно.*

Види контролю поточних знань – усне опитування під час лекцій та лабораторних занять, контрольні роботи, модульний контроль, залік. Вид контролю залишкових знань – тестові завдання.

Оцінювання лабораторних робот:

За лабораторну роботу№1 встановлена максимальна оцінка – 5 балів.
За лабораторну роботу№2 встановлена максимальна оцінка – 5 балів.
За лабораторну роботу№3 встановлена максимальна оцінка – 5 балів.
За лабораторну роботу№4 встановлена максимальна оцінка – 5 балів.
За лабораторну роботу№5 встановлена максимальна оцінка – 6 балів.
За лабораторну роботу№6 встановлена максимальна оцінка – 6 балів.
За лабораторну роботу№7 встановлена максимальна оцінка – 8 балів.

Загальний обсяг навчального часу для денної форми навчання – 120 годин:

лекцій – 30 годин, лабораторних занять – 30 годин, самостійна робота – 60 годин.

**Правила техніки безпеки та охорона праці**

Згідно з «Правилами техніки безпеки в лабораторіях інформатики» студентам забороняється:

- з`являтися та знаходитись приміщенні в нетверезому стані;
- ставити поруч з клавіатурою ємності з рідиною;
- перебувати в приміщенні в верхній одежі та завалювати нею робочі столи та стільці;
- працювати в лабораторії більше 6-ти годин на день (для вагітних жінок – більше 4-х годин);
- за власною ініціативою змінювати закріплені за ними робочі місця та знаходитись в приміщенні під час роботи іншої учбової групи;
- самостійно виконувати вмикання електроживлення лабораторії та заміну складових частин ПК, що вийшли із ладу.

У випадку виявлення несправностей обчислювальної техніки студент повинен сповістити про це викладача чи будь-кого з навчально-допоміжного персоналу лабораторії.

## ЛАБОРАТОРНА РОБОТА №1

**Тема:** Створення XML-документу

**Мета роботи:** розглянути структуру XML-документу, та виконати створення власного XML-документ згідно з правилами створення well-formed XML-документу.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

XML - текстовий формат, призначений для зберігання структурованих даних (замість існуючих файлів баз даних), для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов, іноді званих словниками. XML, які є спрощеною підмножиною мови SGML. Стандартом визначено два рівні правильності документа XML:

Опис XML- документа являє собою простий текст, який можна набрати в будь-якому текстовому редакторі. Наприклад, створіть у текстовому редакторі Notepad новий файл і введіть текст XML- документа та збережіть його з розширенням .xml

Кожен XML- документ розмічається тегами. Тег – це текст, укладений в кутові дужки, який не відноситься до змісту документа, а вказує на початок або кінець будь-якого документа.

***Елемент документа XML*** має початковий тег, що задає ім'я елементу і додаткову інформацію (атрибути), і кінцевий тег, що містить таке ж саме ім'я елемента зі знаком "/" попереду. Елементи визначають логічну структуру документа і несуть в собі інформацію, що міститься в документі. Ім'я елемента вважається так само його типом. Елементи XML-документа можуть бути вкладеними.

Поки не заданий формат відображення XML- документа на екрані, браузер застосовує спосіб, прийнятий за замовчуванням (IE5 буде використовувати вбудовану таблицю стилів для відображення документа). Одним з варіантів вказівки способу відображення документа є створення для нього таблиці каскадних стилів (CSS).

***Правильно оформленими (well-formed) XML-документом*** називається документ, що задовольняє мінімальному набору правил відповідності для XML-документа. *Правильно оформлений XML-документ складається з двох основних частин:* ***прологу і кореневого елемента.*** Крім цього він може містити ***коментарі, інструкції і пропуск***. Кореневий елемент може містити вкладені елементи. Елементи повинні бути правильно вкладені. Якщо елемент розпочинається

всередині деякого іншого елемента, то й закінчуватися він повинен всередині того ж елемента.

Кожен елемент складається з початкового тега, вмісту і кінцевого тега. Винятком є порожній елемент, який може використовувати єдиний тег порожнього елемента <Emptylement />.

***Правила використання імен елементів (типів елемента):***

- ім'я повинне починатися з букви або символу підкреслення (_),
- наступні після першого символи можуть бути буквами, цифри, точкою, тире або підкресленням.
- Не слід використовувати імена, що починаються із префікса "xml" (в будь-якому поєднанні малих або великих літер).
- ім'я, що записане в початковому тегу, має в точності відповідати імені в кінцевому тезі;
- Дотримання регістра істотно для імен елементів, як і для всього тексту в описі розмітки.

***Вмістом елемента*** вважається текст, розташований між початковим і кінцевим тегами. У початковий тег елемента або в тег порожнього елемента можна включити один або кілька описів атрибутів. Опис атрибута являє собою пару ***ім'я = значення***, пов'язану з даним елементом. Кожне ім'я атрибута може тільки один раз бути присутнім в початковому тегу елемента. Правила іменування атрибутів аналогічні правилам іменування елементів. Завдання атрибутів забезпечує альтернативний спосіб включення інформації в елемент. Значення, яке можна привласнювати атрибуту, є рядком символів, обмежених одинарними або подвійними лапками.

***Коментар*** починається з символів ***<!*** – і закінчується символами ***–>***. Усередині коментарів не може міститися подвійне тире (--), символ лівої кутової дужки (<) і знак амперсанда (&). Коментарі можна вставляти в будь-яке місце XML-документа, крім опису тега.

***Інструкції*** (інструкції по обробці) призначені для XML-процесора, який буде обробляти документ. Загальний вигляд інструкції:

***<? найменування дані?>***

***Найменування*** вказує одержувача інструкції. Параметр ***дані*** задає зміст інструкції. За назвою конкретний XML-процесор визначає, призначені дані йому або іншому процесору. Інструкції можуть поміщатися в будь-яке місце XML-документа поза описом тегів.

*Наприклад, інструкція*

```
<?Xml-stylesheet type="text/css"href="file_2.css"?>
```

наказує Internet Explorer використовувати CSS-таблицю з файлу file_2.css.

Усередині символьних даних у вмісті елементу не можна поміщати деякі спеціальні символи (<, &, ...), так як це може призвести до плутанини при обробці документа. Одним з можливих шляхів подолання цих обмежень є використання розділів символьних даних. Такий розділ починається з символів ***<! [CDATA [і закінчується]]>.***

Всі символи всередині розділу CDATA розглядаються як літеральна частина символьних даних елемента, а не як XML-розмітка. Розділ CDATA може розташовуватися в будь-якому місці документа, що займається символьними даними. Розділи CDATA не можуть бути вкладеними.

Документ XML може містити ***порожні рядки***, що складаються з одного або декількох пропусків, символів табуляції, символу Enter. Можна вільно додавати прогалини і переклади рядків між: початковими і кінцевими тегами; коментарями; інструкціями по обробці.

Неповинно бути пропусків між кутовою дужкою що відкривається і іменем о елемента.

## ПРИКЛАД ВИКОНАННЯ РОБОТИ

*Розглянемо структуру XML*

Перший рядок XML-документа називається ***оголошення XML*** – це необов'язковий рядок, який вказує версію стандарту XML (зазвичай це 1.0), також тут може бути вказано кодування символів і зовнішні залежності. Наприклад:

```
<? Xml version = "1.0" encoding = "utf-8"?>
```

Найважливіша обов'язкова синтаксична вимога полягає в тому, що документ має ***тільки один кореневий елемент*** (так само іноді званий *елемент документа*). Це означає, що текст або інші дані всього документа повинні бути розташовані між єдиним початковим кореневих тегом і відповідним йому кінцевим тегом. Наприклад:

```
<Comp> комп'ютер </comp>
```

Інша частина цього XML-документа складається з вкладених елементів, деякі з яких мають атрибути і вміст. ***Елемент*** зазвичай складається з тегів, що відкриваються і закриваються, обрамляють текст і інші елементи.

```
<Video> відеокарта </video>
```

Крім змісту у елемента можуть бути ***атрибути*** – пари *ім'я–значення*, що додаються в тег, що відкривається після назви елемента. Значення атрибутів

завжди розташовані в лапки (одинарні або подвійні), одне і те ж ім'я атрибута не може зустрічатися двічі на одному елементі. Не рекомендується використовувати різні типи лапок для значень атрибутів одного тега. Наприклад:

```
<Video model = "6600"> відеокарта </video>
```

У наведеному прикладі у елемента *«video»* є атрибут: *«model»*, що має значення *«6600»*

У будь-якому місці дерева може бути розміщений елемент-коментар. XML-коментарі розміщуються всередині спеціального тега, який починається з символів *<!* – і закінчується символами –>. Два знака дефіс (--) у коментарі присутнім не можуть бути.

```
<!-- Коментар-->
```

*Наприклад:*

```
    <log>
    <event date=" 27/May/2016:04:22:41 " result="success">
    <ip-from> 192.11.112.8 </ip-from>
    <method>GET</method>
    <url-to> /misc/</url-to>
    <response>200</response>
    </event>
</log>
```

**ЗАВДАННЯ**

Розглянемо приклад виконання завдання. Необхідно створити Xml-документ на основі даної таблиці.

Використовувати виділені слова лівого стовпчика таблиці, як теги, а звичайні – як атрібути. В якості параметрів використовуйте дані правого стовпчика. Збережіть документ з назвою Завдання1.xml

| | |
|---|---|
| **Основні** | |
| Виробник | ASUS |
| Модель | EAH5670/DI/512MD5 |
| Длина | 170 mm |
| **Відео** | |
| Відеопам’ять | 512 Мб |
| RAMDAC | 400МГц, 10 бит на канал |
| Максимальна роздільна здатність 2D/3D | 2560x1600 або 1920x1200 для HDMI |
| Підтримка HDCP | 1080p |
| Підтримка API | DirectX11, OpenGL 3.2 |
| **Підтримка пам’яті** | |
| Тип пам’яті | GDDR5 |
| Розрядність шини пам'яті | 128бит |
| Частота пам'яті | 1000Гц |
| **Конфігурація** | |
| Чіп | Radeon HD 5670 |
| Техпроцес | 40нм |

| | |
|---|---|
| Частота чипу | 775 МГц |
| Кількість піксельних конвейерів | 20,8 блоков выборки текстур |
| Кількість процесорів | 4 |
| **Роз'єми і виходи** | |
| Порти | DVI-I, HDMI,15-піновий конвектор D-Sub |
| **Інтерфейс** | |
| Інтерфейс | PCI Express 2.116x |
| **Охолодження** | |
| Управління швидкістю обертання | Немає |
| Охолодження | Активне |
| Конструкція системи охолодження | Система охолодження сусідній з відеокартою слот |
| **Живлення** | |
| Роз'єми живлення | Немає |
| **Сумісність** | |
| Вимоги до системи | Блок живлення 400 Вт або 500Вт |
| Підтримка обчислень загального призначення | ATI Stream, OpenCL 1.0, DirectCompute 11 |
| Підтримка ОС | Windows XP, Windows 7 |
| **Логістика** | |
| Розміри упаковки | 30.5x22.5x5.7 |
| Вага бруто | 0.755 кг |

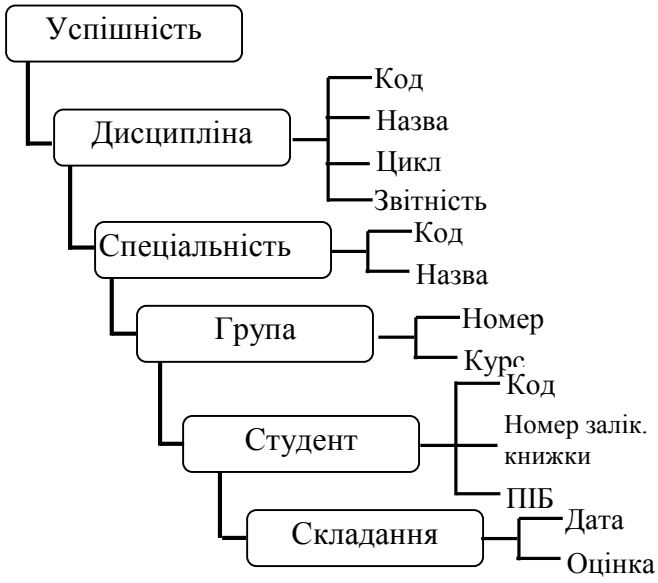
В результаті Ваш створених XML-файл може виглядати наступним чином:

```
<?xml version="1.0" encoding="utf-8"?>
<Harakter>
<Osnovnie virobniki="ASUS " model="EAH6670/DI/512MD5 " dlina=" 170 мм
">
</Osnovnie>
<Video videopamyat="512" RAMDAC="400 МГц, 10 бит на канал."
HDCP="Есть (1080p)" API="DirectX 11 и OpenGL 3.2 ">
</Video>
<Pamyat tip="ddr5" shina="128 біт" chastota="1000 МГц">
</Pamyat>
<Config chip="Radeon HD 5670" chastota_chipa="775 МГц" kol-
vo_processorov="4 ">
</Config>
<Razemi porti="D-sub,dvi">
</Razemi>
<Interface>PCI Express 2.1</Interface>
<Ohlazhdenie upravlenie="net" ohlazhdenie="activ">
</Ohlazhdenie>
<Sovmestimost pitanie="400 Вт" os="Windows XP x64, Windows7 >
</Sovmestimost>
</Harakter>
```

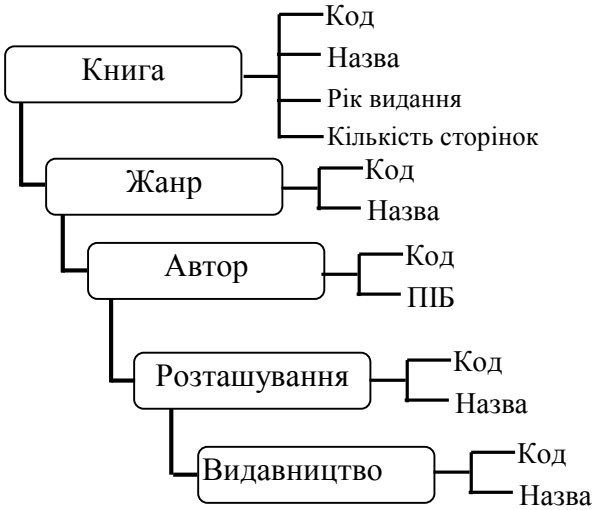
**ВАРІАНТИ ЗАВДАНЬ:**

За наданою схемою на рисунках і обраного варіанту необхідно створити XML-документ. Для кожного елемента першого рівня передбачити не менше 3 примірників, для кожного елемента наступних рівнів – не менше двох примірників.

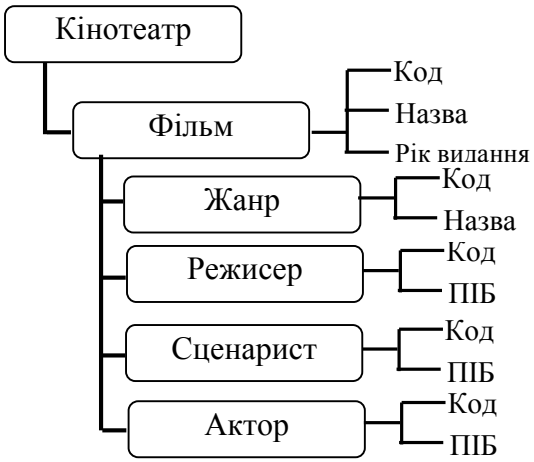
**Варіант №1** Результати сесії студентів



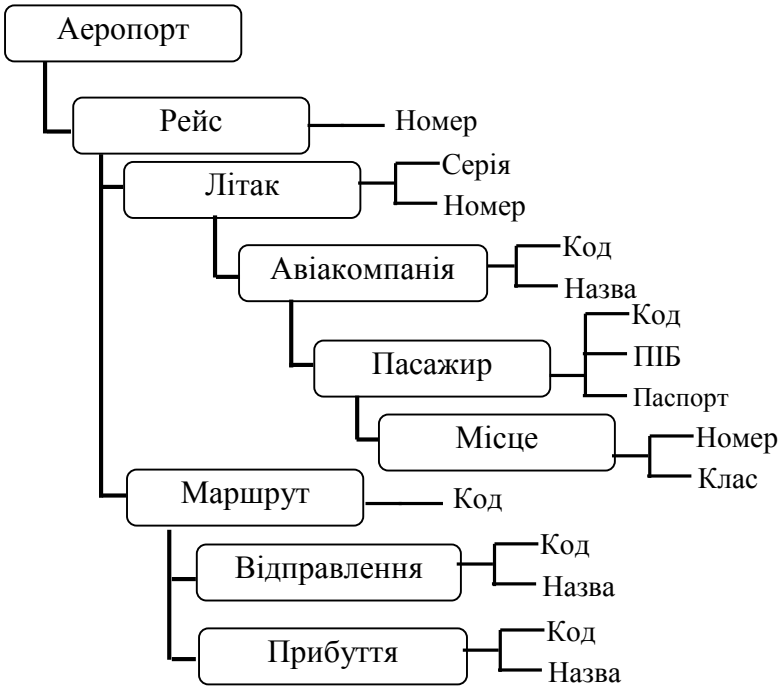

**Варіант №2** Книжковий каталог



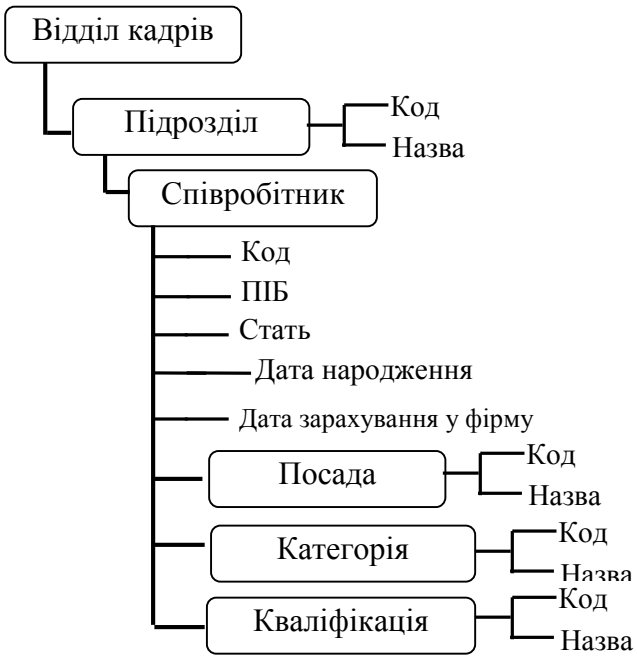

**Варіант №3** Анонс репертуару кінотеатру



**Варіант №4.** Інформаційне табло аеропорту




**Варіант № 5.** Облік співробітників фірми в відділі кадрів




**Варіант № 6.** Записна книжка адресів

**Варіант № 7.** Розклад занять в університеті

**Варіант № 8.** Облік товарів на складі

**Варіант № 9.** Відділ кадрів студентів

**Варіант № 10.** Каталог загальноосвітніх Інтернет-ресурсів

## КОНТРОЛЬНІ ПИТАННЯ

1. Що являє собою XML-документ?
2. Для чого використовується XML?
3. З яких розділів складається XML-документ?
4. Яку інформацію містить декларація XML-документа?
5. Яка інформація розміщується в пролозі XML-документа?
6. Як будуються елементи XML-документа?
7. Які типи елементів ви знаєте?
8. Що таке атрибути?
9. Які синтаксичні правила створення атрибутів XML- документа?
10. Для чого в XML-документах використовуються коментарі?
11. Для чого в XML-документі необхідно простір імен?

## ЛАБОРАТОРНА РОБОТА №2

**Тема:** створення XML-документа і його відображення з допомогою каскадних таблиць стилів CSS.

**Мета роботи**: вивчити основи застосування CSS при формуванні XML-документу. Навчитися відображати XML-документи з використанням каскадної таблиці стилів

## ТЕОРЕТИЧНІ ВІДОМОСТІ

Відображення XML-документів з використанням каскадної таблиці стилів здійснюється в два етапи: створення файлу таблиці стилів і зв'язування таблиці стилів з XML-документом.

Таблиця стилів складається з одного або декількох правил, іноді їх називають набором правил. Правило містить інформацію по відображенню певного типу елемента в XML-документі. Селектор являє собою ім'я типу елемента, до якого відноситься інформація для відображенню.

Каскадні таблиці стилів CSS не чутливі до регістру. Набір успадкованих властивостей, які присвоєні певному елементу, діє на всі дочірні елементи, прямо або побічно вкладені в нього, якщо тільки вони не переустановлюються згодом для певного дочірнього елемента.

Для неуспадкованих властивостей, якщо не задано значення властивості для конкретного елемента, браузер використовує значення властивості за замовчуванням.

***Використання контекстуальних селекторів***

***Контекстуальним селектором (contextual)*** називається селектор, в якому ім'я елемента може передувати іменам одного або декількох елементів-предків (батьківський, батьківський плюс батьківський батька і т.п.). У цьому випадку правило буде застосовано тільки до елементів з цим ім'ям, які є вкладеними подібним чином. Між іменами елементів в контекстному селекторі ставлять тільки пробіли.

***Родовим селектором (generic)*** називається селектор, який не включає імен елементів-предків.

Якщо певна властивість для одного і того ж елемента має одну установку в правилі з контекстуальних селектором, і іншу установку в правилі з родовим селектором, установка в правилі з контекстуальних селектором домінує, оскільки є більш конкретизована.

Наприклад, фрагмент програмного коду XML-документа, де <MAPS> – кореневий елемент:

```
<MAPS>
<CITY>
<STATE>California</STATE>
</MAPS>
```

Правила приєднаної таблиці стилів: CITY STATE

```
{font-style:normal}
STATE
font-style:italic}
<NAME>Santa Fe</NAME>
<STATE>New Mexico</STATE>
</CITY>
```

В результаті браузер відобразить «New Mexico» звичайним шрифтом, а «California» – курсивом.

***Використання атрибуту STYLE***

Можна в XML-документі використовувати спеціальний атрибут STYLE, замість того, щоб встановлювати одне або кілька певних властивостей окремого елемента в таблиці стилів.

Якщо значення властивості, встановленої за допомогою атрибута STYLE, конфліктує із значенням властивості, встановленого в таблиці стилів, установка за допомогою атрибута STYLE має пріоритет. Таким чином, атрибут STYLE є зручним засобом, щоб перевстановити для певного елемента значення властивості, присвоєне для типу елемента в приєднаної таблиці стилів.

Використання атрибуту STYLE порушує принцип CSS щодо зберігання інформації про форматування окремо від визначення вмісту документа і структури XML-файла. Щоб встановити одне або кілька значень властивостей, оголошення в значення атрибуту STYLE включається у вигляді рядка, укладеного в лапки, відокремлюючи індивідуальні оголошення крапкою з комою. Наприклад,

```
 <TITLE STYLE='font-style:normal;
     font-size:14pt'> The Adventures of Huckleberry Finn
       TITLE/>
```

***Імпорт інших таблиць стилів***

Можна за допомогою директиви @import в одну каскадну таблицю стилів вбудувати одну або кілька інших таблиць стилів. Можливість імпорту окремих таблиць стилів дозволяє зберігати правила для пов'язаних стилів в окремих файлах, а потім об'єднувати їх при створенні документів певного типу. Директива @import

повинна розташовуватися на початку таблиці стилів перед правилами. На початку таблиці стилів можна розмістити кілька директив @import.

```
@import url (URL ІмпортТаблСтіл);
```

Наприклад

```
/* Файл: Inventory01.css */ @import url (Book.css);
     BOOK
     {Display: block; margin-top: 12pt; font-size: 10pt}
/* Продовження таблиці стилів ... */
```

У разі виникнення конфлікту правил основна таблиця стилів (з файлу, в який здійснюється імпорт) має пріоритет над імпортованими таблицями стилів. Якщо імпортуємо кілька таблиць стилів, правила з таблиці стилів, імпортованої останньою, мають пріоритет над правилами з раніше імпортованих таблиць стилів.

***Зв'язування таблиці стилів з XML-документом***

Щоб зв'язати таблицю каскадних стилів з XML-документом, необхідно вставити в документ зарезервовану інструкцію по обробці `xml-stylesheet`. Ця інструкція з обробки має наступну узагальнену форму записи, де URLТаблСтіл є URL, що задає місцезнаходження файлу таблиці стилів:

```
<? Xml-stylesheet type = "text / css" href = "URLТаблСтіл"?>
```

Можна використовувати повний URL, наприклад:

```
<? Xml-stylesheet type = "text / css" href = "http://www.my_domain.com/Inventory01.css"?>
```

Найчастіше використовується частковий URL, який задає місцезнаходження щодо місцезнаходження XML-документа, що містить інструкцію з обробки `xml-stylesheet`, наприклад:

```
<? Xml-stylesheet type = "text / css" href = "file_2.css"?>
```

Зазвичай інструкція по обробці xml-stylesheet додається в пролог XML-документа, слідом за оголошенням XML. Якщо браузер не може знайти файл таблиці стилів, заданий в інструкції по обробці xml-stylesheet, він відобразить текст документа з використанням свого власного набору властивостей (наприклад, з поточними значеннями гарнітури і розміру шрифту).

Можливо включити в XML-документ більш однієї таблиці стилів, вставивши для кожної з них інструкцію по обробці `xml-stylesheet` на початку XML-документа, наприклад:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="Book01.css"?>
<?xml-stylesheet type="text/css" href="Book02.css"?>
<INVENTORY>
<!—вміст елемента Документ -->
</INVENTORY>
```

Можливість зв'язування з декількома таблицями стилів дозволяє зберігати групи взаємопов'язаних правил в окремих файлах, а потім об'єднувати їх при створенні документів певних типів.

***Пріоритет правил каскадних таблиць стилів***

У таблицях каскадних стилів можна присвоювати значення властивостям на декількох різних рівнях. Основні рівні, на яких можна привласнювати значення властивості – від вищого рівня пріоритету до нижчого.

Значення властивості в атрибуті STYLE для певного елемента в XML-документі, має найвищий пріоритет. Якщо не встановлено властивість в атрибуті STYLE, браузер використовує значення властивості, оголошеного в правилі CSS з контекстуальних селектором. Якщо не оголошено значення певної властивості в правилі, що має відповідний контекстуальний селектор, браузер використовує значення, оголошене в правилі з родовим селектором (тобто селектором, який включає тільки ім'я елемента). Якщо не оголошено значення для певного властивості для елементу в родовому правилі, браузер використовує установку властивості, оголошену для найближчого елемента-предка (батька, батька батька і т.д.).

Якщо таблиця стилів не містить установку властивості для будь-якого батьківського елемента, браузер використовує свою власну установку. Такий установкою може бути значення за замовчуванням, вбудоване в браузер, або значення, задане користувачем браузера. Наприклад, якщо не встановлено значення для властивості `font-family`, браузер використовує своє власне значення цієї властивості для відображення всіх елементів (В Internet Explorer це шрифт Times New Roman, якщо тільки користувач браузера не вибере інше сімейство шрифтів, скориставшись командою Internet Options – Властивості оглядача, з меню Tools (Сервіс).

Якщо для певного властивості встановлені конфліктуючі значення на одному і тому ж рівні, то в такому випадку браузер використовує останню установку, яку він обробив.

Цей процес має місце тільки для успадкованих властивостей. Для неуспадкованих властивостей браузер використовує значення властивості за замовчуванням.

***Властивості CSS, які часто використовуються в XML-документах***

*Установка властивості* `display`

Властивість `display` керує основним способом відображення тексту елемента браузером. Можливо призначити йому три ключових слова CSS. Властивість `display` не успадковується дочірніми елементами. Можливі значення властивості:

`Block` – браузер завжди поміщає новий рядок перед і після тексту елемента (який включає і текст, що належить будь-яким дочірнім елементам). В результаті текст елемента відображається в окремому "блоці" з попереднім текстом вище і подальшим текстом нижче. Присвоєння значення `block` дозволяє форматувати текст із застосуванням різних властивостей обрамлення до блоку тексту, таких як поля, рамки і відступи. Елемент `block` схожий на абзац, який відділений пробілами від попереднього і наступного тексту, і для якого можна задавати відступи, рамки і т.д.;

`inline` (за замовчуванням) – браузер не вставляє новий рядок перед або після тексту елемента (якщо тільки попередній текст досяг правої межі вікна, і браузер повинен перенести текст на наступний рядок). Він буде вставляти перенесення рядка всередині тексту елемента тільки з метою вмістити текст у вікні. Текст елемента, таким чином, може бути розміщений в тому самому рядку, що і попередній або наступний текст. Елемент `inline` аналогічний групі символів всередині абзацу в програмі текстового процесора;

`none` – браузер не відображає елемент. Можливо використовувати цю установку для елементів, що несуть інформацію, яку не бажано поміщати на екрані.

***Установка властивостей шрифту***

У стандартній CSS-таблиці передбачені наступні властивості, що визначають вид шрифту, який використовується для відображення тексту елемента:

```
font-family ;
font-size ;
font-style ;
font-weight ;
font-variant.
```

Всі ці властивості успадковуються дочірніми елементами.

***Установка властивостей фону***

Стандарт CSS підтримує такі властивості, що дозволяють модифікувати фонове оформлення елемента:

```
background-color;
background-image;
background-repeat;
```

background-position.

Технічно дочірні елементи не успадковують властивостей фону. Однак за замовчуванням фон елемента є прозорим. Це означає, що якщо опустити всі властивості фону для дочірнього елемента, буде видно колір фону або малюнок батьківського елемента (або браузера), тобто фон дочірнього елемента буде таким же, що і фон батьківського елемента.

***Установка властивостей розбивки тексту і вирівнювання***

Стандарт CSS підтримує такі властивості, що дозволяють модифікувати розбивку, вирівнювання та інші характеристики тексту:

letter-spacing ;
vertical-align ;
text-align ;
text-indent ;
line-height ;
text-transform ;
text-decoration.

Дочірні елементи успадковують всі ці властивості, за винятком vertical-align.

***Установка властивостей текстових областей***

Специфікація CSS підтримує ряд властивостей (типу box), які можна використовувати для форматування блоку тексту, що належить елементу. До цих властивостей відносяться наступні:

– властивість margin додає невидиме (прозоре) поле навколо елемента, зовні від видимої рамки (якщо вона є);

– властивість border відображає видиму рамку певного стилю навколо елемента;

– властивість padding додає просвіт безпосередньо зовні від вмісту елемента, але всередині рамки (якщо вона є);

– властивості завдання розмірів height і width встановлюють розміри області вмісту елемента з доданими просвітом і рамкою;

властивості завдання позицій float і clear встановлюють положення елемента щодо оточуючих елементів.

Елемент block – це елемент, для властивості display з встановленим значенням block, а елемент inline – це елемент, для властивості display з встановленим значенням inline. В Internet Explorer перші три групи властивостей (margin, border і padding) діють тільки на елементи block. І inline.

***Установка властивостей управління полями***

За замовчуванням ширина полів навколо елемента дорівнює нулю. Щоб додати поле з однією або з декількох сторін елемента, можна привласнити нульове значення одному або декільком з наступних властивостей: `margin-top`;

`margin-right;`

`margin-bottom;`

`margin-left.`

Можливо використовувати стенографічні властивості. Установка властивостей управління обрамленням. Для створення видимого обрамлення навколо елемента можна скористатися наступними властивостями CSS:

`border-style;`

`border-width;`

`border-color.`

***Установка властивостей просвіту між обрамленням і текстом***

Властивість просвіт (`padding`) додають простір безпосередньо навколо вмісту елементу, всередині від наявної навколо елемента рамки. Без просвіту кордону рамки розташовуються безпосередньо поблизу тексту елемента. Додавання просвіту покращує сприйняття рамки. За замовчуванням ширина просвіту для елемента встановлюється близькою до нуля. Щоб додати просвіт з одного або з декількох сторін від тексту елемента, можна привласнити нульове значення одному або декільком з наступних властивостей:

```
padding-top;
padding-right;
padding-bottom;
padding-left.
```

Можливо встановлювати для цих властивостей значення в будь-яких розмірних одиницях, що допускаються в CSS.

Як і для вмісту елемента, для просвіту відображається будь-фон у вигляді суцільної заливки або малюнків, які призначені для елемента. На відміну від області полів, для яких відображається фон батьківського елемента.

## ПРИКЛАД ВИКОНАННЯ РОБОТИ

Опрацюйте нижче запропонований приклад створення XML-документа і його відображення з допомогою каскадних таблиць стилів CCS.

Створіть у текстовому редакторі Notepad новий файл і введіть текст XML-документа, зберігши з розширенням .xml.

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

```
     <STUDENT>
               <FAMILI>Сидоров</FAMILI>
               <NAME>Иван</NAME>
               <YEAR>2005</YEAR>
               <GROUP>КН-19</GROUP>
     </STUDENT>
     <STUDENT>
               <FAMILI>Иванова</FAMILI>
               <NAME>Клавдия</NAME>
               <YEAR>2003</YEAR>
               <GROUP>КН-18</GROUP>
     </STUDENT>
     <STUDENT>
               <FAMILI>Журавлева</FAMILI>
               <NAME>Анастасия</NAME>
               <YEAR>2004</YEAR>
               <GROUP>КН-31</GROUP>
     </STUDENT>
     <STUDENT>
               <FAMILI>Семенников</FAMILI>
               <NAME>Антон</NAME>
               <YEAR>2006</YEAR>
               <GROUP>КН-20</GROUP>
     </STUDENT>
</zavdannya>
```

Даний документ складається з двох основних частин: прологу і кореневого документа (званого також елементом документа). Елемент документа називається тут zavdannya, його початковий тег – <zavdannya>, а кінцевий – </zavdannya>, а вміст – чотири вкладених елемента STUDENT. У свою чергу кожен елемент STUDENT містить ряд вкладених елементів.

Відкрийте документ за допомогою браузера. Після перевірки синтаксису, документ відобразиться на екрані. При наявності помилок замість документа на екран буде видано повідомлення про неможливість відобразити сторінку.

Спробуйте змінити ступінь деталізації представлення елементів документа. Клацніть на символі знака мінус (-) зліва від початкового тега, щоб згорнути елемент, або на знаку плюс (+) поруч із згорнутим елементом, щоб розгорнути його.

Створіть у файлі file.css каскадну таблицю:

```
STUDENT
     {display: block;
     margin-top: 12pt;
     font-size: 10pt}
FAMILY
     {font-style: italic}
NAME
 {font-weight: bold}
```

Відкрийте в текстовому редакторі файл, створений в першому пункті завдання, і другим рядком документа наступну інструкцію по обробці:

```
<?xml version="1.0" encoding="utf-8"?>
      <?xml-stylesheet type=”text/css” href=”file.css”?>
     <! -- Ім’я файлу: fale_10.xml -->
<zavdannya>
     <STUDENT>
```

**ВАРІАНТИ ЗАВДАНЬ:**

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Створений XML-документ повинен відповідати наступним вимогам:

- документи повинні мати глибину вкладеності не менше чотирьох елементів;
- число елементів документа, які не мають вкладених, має бути не менше п'яти;
- елементи документа повинні містити коментарі про свій зміст;
- документ повинен включати елементи, що містять символьні дані і дочірній елементи.

Якщо необхідно доробити до вимог цей документ, – внесіть зміни.

Створіть таблицю каскадних стилів, яка отформатує створений XML-документ. Створена CSS-таблиця повинна відповідати наступним правилам:

- CSS-таблиця повинна включати як контекстуальні, так і родові селектори;
- дочірні елементи повинні наслідувати CSS-формат батьківського елемента;
- створена CSS-таблиця повинна імпортувати іншу таблицю стилів;
- таблиця стилів повинна включати використання атрибута STYLE;
- створена таблиця стилів повинна включати:
    - властивості шрифту;
    - властивість фону;
    - властивості розбивки і вирівнювання тексту;
    - властивості текстових областей;
    - властивості полів і кордонів;
    - псевдо елементи.

У звіті до виконання лабораторної роботи №2 необхідно представити код xml-документа, код таблиць CSS та скріншот табличного представлення документа.

## КОНТРОЛЬНІ ПИТАННЯ

1. Визначте мету застосування каскадних таблиць стилів CSS для відображення XML-документу.

2. Сформулюйте основні правила складання таблиць стилів.

3. Зазначте умови використання контекстуальних селекторів.

4. Зазначте умови використання родового селектора generic.

5. Переваги використання атрибуту STYLE у каскадних таблицях стилів.

6. Сформулюйте Пріоритет правил каскадних таблиць стилів.

7. Застосування механізму імпорту інших таблиць стилів.

8. Яким чином зв'язуються таблиці стилів з XML-документом?

9. Перелічите властивості CSS, які найчастіше використовуються в XML-документах.

10. Переваги та недоліки використання аскадних таблиць стилів CSS для відображення XML-документу.

## ЛАБОРАТОРНА РОБОТА №3

**Тема:** мова схем DTD та створення DTD-схеми для XML-документа.

**Мета роботи**: вивчити мову схем DTD та навчитися створювати DTD-визначення для побудови DTD-схеми XML-документу.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

DTD (Document Type Definition – визначення типу документа) включає в себе два поняття: термін, який використовується для опису схеми документа або його частини мовою схем DTD та мова схем DTD (DTD schema language) – штучна мова, яка використовується для запису фактичних синтаксичних правил метамовою розмітки тексту SGML і XML. З моменту її впровадження інші мови схем для специфікацій, такі як XML Schema і RELAX NG, випускаються з додатковою функціональністю.

***Оголошення елементів***

Оголошення елементів утворюють перелік дозволених назв елементів в документі, а також визначають інформацію щодо тегів (чи є вони обов'язковими) і моделі вмісту для кожного елемента.

Різні ключові слова і символи визначають вміст елемента:

EMPTY – пусте вміст

ANY – будь-який вміст

, – вказує порядок

| – поділ альтернатив

() – угруповання

* – будь-яка кількість елементів (нуль і більше)

+ – принаймні один елемент (один і більше)

? – необов'язкове наявність елемента (нуль або один)

Якщо немає *, + або? – елемент повинен бути тільки один

```
<! ELEMENT HDD - - (250 | 500) +>
```

Елемент HDD повинен містити один і більше елементів 250 або 500 в довільному порядку.

***Визначення атрибутів***

З кожним елементом DTD-документа можна зіставити список атрибутів. Для цього використовується директива! `ATTLIST`, в якій зазначаються ім'я елемента, з яким може бути зіставлений список атрибутів і параметри кожного атрибута: його ім'я, тип і властивості за замовчуванням.

Наприклад:

```
<! ATTLIST Video model CDATA #REQUIRED>
```

У цьому прикладі визначено атрибут Video для елемента model. Він є обов'язковим.

*Існують такі типи атрибутів:*

- CDATA (Character set of data) – значенням атрибута можуть бути будь-які символьні дані;
- ID – значенням атрибута повинен бути унікальний ідентифікатор елемента;
- IDREF – значенням елемента є посилання на елемент по його ID;
- IDREFS – теж що і IDREF, але з можливістю посилань не по одному ідентифікатору, а за кількома;
- NMTOKEN – значенням атрибута може бути послідовність символів, в чомусь схожа з ім'ям (звідси і назвою – name token). Це рядок, який містить будь-яку комбінацію тих символів, які дозволено використовувати для імен XML;
- NMTOKENS – значенням атрибута є список значень;
- ENTITY – значення використовується для посилання на зовнішню сутність;
- ENTITIES – дозволяє задати список зовнішніх сутностей, розділених пробілами;
- NOTATION – значенням атрибута може бути одна з раніше визначених нотацій;
- NOTATIONS – дозволяє задати список нотацій Listings і NOTATION–listings;
- ENUMERATION – задає список можливих альтернатив значень;

*За замовчуванням існують наступні властивості:*

- IMPLIED – значення атрибута вказувати не обов'язково;
- REQUIRED – значення атрибута обов'язково повинно бути зазначено;
- FIXED – значення цього атрибута задано як константа в DTD і в документі не може бути змінено;
- деяке конкретне значення, яке використовується за умовчанням.

Наприклад:

```
<?xml encoding="koi8-r"?>
<!ELEMENT log (event)+>
<!ELEMENT event (ip-from,method,uri-to,result)>
<!ELEMENT method (#PCDATA)>
<!ELEMENT ip-from (#PCDATA)>
<!ELEMENT url-to (#PCDATA)>
<!ELEMENT response (#PCDATA)>
<!ATTLIST event
result CDATA #IMPLIED
```

```
date CDATA #IMPLIED>
```

**ПРИКЛАД ВИКОНАННЯ РОБОТИ**

Розглядаючи приклад виконання у лабораторній роботі №1 ми створили XML-документ під назвою Завдання1.xml і переконалися, що набір використуваних при цьому тегів що дозволяє здійснювати будь-які маніпуляції з нашою інформацією. В такому випадку, для того, щоб затвердити правила нашої нової мови, тобто список допустимих елементів, їх можливий вміст і атрибути, ми повинні створити DTD - визначення.

Необхідно створити dtd- файл на основі таблиці (див. лабораторну роботу №1) під ім'ям Завдання1.dtd і включіть в XML-документ новий рядок

```
<! DOCTYPE VIDEO SYSTEM " Завдання1.dtd">
```

В результаті документ Завдання1.dtd може виглядати наступним чином:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Enter you external DTD here -->
<!ELEMENT Haracter (Osnovnie,
Video,Pamyat,Config,Razemi,Interface,Ohlazhdenie,Sovmestimost)+ >
<!ELEMENT Osnovnie (#PCDATA)>
<!ATTLIST Osnovnie
proizvoditel CDATA #REQUIRED
model CDATA #REQUIRED
dlina CDATA #REQUIRED >
<!ELEMENT Video (#PCDATA)>
<!ATTLIST Video
videopamyat (256|512|1024) #REQUIRED
RAMDAC CDATA #REQUIRED
HDCP CDATA #REQUIRED
API CDATA #REQUIRED >
<!ELEMENT Pamyat (#PCDATA)>
<!ATTLIST Pamyat
tip (ddr2|ddr3|ddr5) #REQUIRED
shina (128|256|448) #REQUIRED
chastota CDATA #REQUIRED >
<!ELEMENT Config (#PCDATA)>
<!ATTLIST Config
chip CDATA #REQUIRED
chastota_chipa CDATA #REQUIRED
kol-vo_processorov CDATA #REQUIRED >
<!ELEMENT Razemi (#PCDATA)>
```

```
<!ATTLIST Razemi
porti CDATA #REQUIRED >
<!ELEMENT Interface (#PCDATA)>
<!ELEMENT Ohlazhdenie (#PCDATA)>
<!ATTLIST Ohlazhdenie
upravlenie (da|net) #FIXED "net"
ohlazhdenie (activ|passiv) #REQUIRED>
<!ELEMENT Sovmestimost (#PCDATA)>
<!ATTLIST Sovmestimost
pitanie CDATA #REQUIRED
os CDATA #REQUIRED>
```

В результаті Ваш XML-файл може виглядати так:

```
<?xml version="1.0" encoding="utf-8"?>
<Harakter>
<Osnovnie proizvoditel="ASUS " model="EAH5670/DI/512MD5 " dlina="
170 мм ">
</Osnovnie>
<Video videopamyat="512" RAMDAC="400 МГц, 10 бит на канал."
HDCP="Есть (1080p)" API="DirectX 11 и OpenGL 3.2 ">
</Video>
<Pamyat tip="ddr5" shina="256" chastota="1000 МГц">
</Pamyat>
<Config chip="Radeon HD 5670"
chastota_chipa="775 МГц" kol-
vo_processorov="400 ">
</Config>
<Razemi porti="D-sub,dvi">
</Razemi>
<Interface>PCI Express 2.1</Interface>
<Ohlazhdenie upravlenie="net" ohlazhdenie="activ">
</Ohlazhdenie>
<Sovmestimost pitanie="400 Вт" os="Windows XP x64, Windows XP,
Windows MCE 2005, Windows Vista, Windows 7">
</Sovmestimost>
</Harakter>
...
```

Проверить все ли данные внесены в XML-файл

Сделайте отчёт о проделанной работе, включив в него скриншоты загрузки указанных файлов в различных браузерах (IE, Opera, Google Chrome и т.п.)

**ВАРІАНТИ ЗАВДАНЬ:**

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Створіть dtd-файл на основі схеми (див. завдання до лабораторної роботи №1) створіть файл з розширенням `****.dtd` і включіть в XML-документ необхідний рядок .

У звіті до лабораторної роботи необхідно надати лістинг вашого dtd- файлу і результат відображення вашого XML-файлу різними браузерами.

## КОНТРОЛЬНІ ПИТАННЯ

1. Для чого служить DTD схема?
2. Яким чином в DTD-схемІ задаються елементи і атрибути?
3. Що можуть містити елементи?
4. За допомогою яких ключових слів задається зміст елементів? Значення атрибутів?
5. За допомогою якого символу в DTD-схемі задається необов'язкове входження елемента «book»?
6. Як визначається обов'язкове входження елемента і входження елемента що повторюється?
7. Як визначається послідовне входження членів списку?
8. Які типи атрибутів найбільш часто використовують у документі?

## ЛАБОРАТОРНА РОБОТА №4

**Тема:** Відображення XML-документа за допомогою мови XSL.

**Мета роботи**: Вивчити конструкції мови XSL. Навчитися відображати XML-документи з використанням мови XSL.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

Мова XSL є додатком XML, так що фактично таблиця стилів (тобто таблиця трансформацій) являє собою документ XML. У зв'язку з цим документ може починатися з декларації XML, яка б показала аналізатору, якою версією мови документ закодований. Кореневим елементом нашої таблиці стилів є елемент

```
<Xsl:stylesheet>:
<Xsl:stylesheet version== "1.0"
xmlns:xsl=="/:1999/XSL/Transform">.
```

Першим атрибутом цього елемента служить версія XSLT, другим – атрибут xmlns: xsl, що містить простір імен для рекомендації трансформації XSL. Цей атрибут декларує простір імен XSLT. З даними простором пов'язаний префікс xsl, так що кореневим елементом фактично є елемент <stylesheet>, але він кваліфікований префіксом простору імен xsl:. Після оголошення простору імен будь-який елемент, що починається з префікса xsl:, входить до складу словника XSL.

Елемент <stylesheet> містить три шаблони, кожен з яких вкладено в елемент <template>. У таблиці стилів цей елемент фактично називається <xsl: template>, так як ми включили простір імен. У елемента <template> є атрибут match, значенням якого є зразок (pattern) в формі вираженія XPath. З ним порівнюється вузол дерева, до якого застосовується шаблон.

Перш за все, процесору XSL треба повідомити бажану форму виведення. Дізнавшись очікуваний формат виведення, процесор почне досліджувати вихідний документ з кореневого вузла.

Перш ніж зробити перетворення дерева документа XML, з нього слід вибрати ті вузли, які піддадуться тому чи іншому перетворенню. Їх можна вибирати по імені, вмісту, атрибутам і іншими ознаками. Умови відбору вузлів задаються зразком (pattern), записаним у вигляді одного або декількох виразів мови XPath 2.0. Вирази, що містяться в зразку, об'єднуються вертикальної рисою |, що означає, що обраний вузол повинен задовольняти хоча б одній висловом зразка. Замість вертикальної риси можна записувати слово union.

У зразку можна записати не всякий вираз XPath, а тільки шлях, кожен крок якого визначається віссю, причому допускаються тільки три осі: child (за замовчуванням), // і attribute. Це означає, що, перебуваючи в якомусь вузлі, ми можемо "побачити", крім нього самого, тільки його атрибути і вузли-нащадки. Зверніть увагу на те, що явний запис осі descendant-or-self неприпустимо, застосовується тільки скорочений запис //. Запис осі attribute: можна скоротити до однієї "собачки" @, а поточний вузол дуже часто позначається крапкою.

Хоча зразок і будується як вираз мови XPath, але його мета – не отобрать послідовність вузлів, а перевірити відповідність вузла даного зразка. Можна сказати, що деякий вузол Node відповідає деякому зразку pattern тоді і тільки тоді, коли вузол Node належить послідовності вузлів – результату обчислення виразу // Pattern.

Наприклад, зразком person // street будуть відповідати всі вузли з послідовності // person // street, а саме всі вузли street, вкладені в вузли-елементи person навіть через кілька промежуточних вузлів.

Треба відразу ж зазначити, що перераховані обмеження стосуються тільки образців. В інших конструкціях мови XSLT можна застосовувати вирази XPath в повному обсязі.

Мова XSLT оголошує близько півсотні елементів. З них 17 елементів верхнего рівня, які можуть бути безпосередньо вкладені в кореневий елемент xsl: stylesheet таблиці стилів. Вони називаються *деклараціями*.

Це елементи xsl:

`import, xsl: include, xsl: attribute-set, xsl: character-map, xsl: date-format, xsl: decimal- format, xsl: function, xsl: import-schema, xsl: key, xsl: namespace-alias, xsl: output, xsl: param, xsl: preserve-space, xsl: sort-key, xsl: strip-space, xsl: template, xsl: variable.`

Більш того, всі ці елементи, крім xsl: param і xsl: variable, можна записувати тільки на верхньому рівні вкладеності, безпосередньо в кореневому елементі xsl: stylesheet.

Крім елементів верхнього рівня вкладеності, в мові XSLT оголошено більш тридцяти елементів, які можна записувати в тілі елементів верхнього рівня.

*Найбільш часто застосовуються елементи xsl:* `apply-templates, xsl: value-of, xsl: copy-of, xsl: sort, xsl: text.`

З усіх елементів XSLT не верхнього рівня вкладеності виділяються інструкції. Не плутайте інструкції XSLT і інструкції по обробці XML. Формально

інструкції XSLT визначаються як елементи, які можна вставляти в конструктор послідовності. До інструкцій, зокрема, відносяться елементи, що створюють вузли всіх семи видів і послідовності вузлів.

Це елементи xsl: `element, xsl: attribute, xsl: text, xsl: comment, xsl: processing-instruction, xsl: namespace, xsl: result-document, xsl: sequence.`

Крім них, *інструкціями* вважаються елементи xsl: `apply-templates, xsl: value-of, xsl: variable` і ін., Всього більше двадцяти елементів. До інструкцій відносяться як елементи, що управляють вибором правил перетворення xsl: `if, xsl: for-each, xsl: choose,` так і елементи, що копіюють вузли `xsl: copy, xsl: copy-of.`

Всі елементи необов'язкові і можуть розташовуватися в будь-якому порядку, за одним винятком: декларації xsl: import, якщо вони є, повинні бути записані первимі. Розглянемо деякі найбільш часто застосовуються елементи XSLT.

***Декларація xsl: variable***

Елемент xsl: variable визначає ім'я об'єкта. Воно записується обов'язковим атрибутом name. Ім'я об'єкта має бути уточненими ім'ям XML типу QName. Крім того, атрибутом select змінної можна задати сам об'єкт, а атрібутом as визначити тип об'єкта. наприклад:

```
<Xsl: variable name="var1" select="count(//person)" as="xs: integer"/>
```

Ім'я var1 буде зберігати число елементів person. Об'єкт може бути отриманий з вмісту елемента xsl: variable:

`<Xsl: variable name="var2">10</xsl: variable>` або створений конструктором послідовності: `<xsl: variable name = "var3">`

`<Xsl: value-of select="count (// person)"/> <xsl:variable>`

Якщо об'єкт не отримано з атрибута select або вмісту елемента xsl: variable, то за замовчуванням ім'я пов'язується з нового рядка.

Для того щоб отримати об'єкт, пов'язаний з ім'ям, певним елементом xsl: variable, перед ім'ям треба поставити знак долара: $ varl, $ var2. При цьому слід враховувати область дії імені. Область дії імені простягається на весь елемент, в якому воно определено, починаючи з місця визначення і включаючи вкладені елементи, якщо тільки в них не визначено те ж саме ім'я.

Імена, визначені безпосередньо в кореневому елементі xsl: stylesheet, називаються *глобальними іменами*, решта – *локальними іменами*.

Хоча слово "variable" і перекладається з англійської мови як "змінна", ім'я, створене елементом xsl: variable, – це не ім'я змінної, його значення не можна змінити. Це тільки назва деякого об'єкту, яке зручно використовувати в тих випадках, коли об'єкт треба використовувати в багатьох місцях таблиці стилів, а його обчислення складно або громіздко.

***Декларація xsl: param***

Елемент xsl: param записується або безпосередньо в елементі xsl: stylesheet, щоб задати параметр перетворення, або в елементі xsl: template, щоб задати параметр правила, або в елементі xsl: function як аргумент функції. У нього один обов'язковий атрибут name, який визначає ім'я параметра. Крім нього, часто присутня необов'язковий атрибут select, в якому записується вираз для отримання значення параметра:

```
<Xsl: param name="pl" select="10+20"/>
```

Якщо атрибут select відсутній, то значення параметра береться з содержімого елемента, яким може бути конструктор послідовності вузлів і атомарних значень:

```
<Xsl: param name="p2">10</xsl: param>
```

Якщо відсутній і атрибут select, і вміст елемента, то параметр получает значення порожнього рядка. Для отримання значення параметра треба записувати його ім'я зі знаком долара: & p1, & p2. наприклад:

```
<Xsl: when test="&p1=10">
```

Правила, що визначають область видимості параметрів, такі ж, як і у імен об'єктів, визначених декларацією xsl: variable.

Ще один необов'язковий атрибут as містить бажаний тип, до якого буде приведено значення параметра.

Нарешті, останній атрибут required, що приймає значення YES або NO за замовчуванням, вказує обов'язковість параметра. Якщо параметр обов'язковий, required = "yes", то елемент xsl: param повинен бути порожнім і не містити атрибут select. В такому випадку він отримає певне значення при виконанні функції або елементом xsl: with-param при виклику шаблону.

***Елемент xsl: with-param***

Елемент xsl: with-param посилається на деякий параметр, ім'я якого записано в обов'язковому атрибуті name. Необов'язковим атрибутом select можна задати вираз, результат обчислення якого буде новим значенням параметра:

```
<Xsl: with-param name="pl" select="100*20"/>
```

Нове значення можна задати і по змісту елемента:

```
<Xsl: with-param name="pl">100</xsl:with-param>
```

Елемент xsl: with-param використовується тільки в інструкціях `xsl: apply-templates, xsl: apply-imports, xsl: call-template.`

***Інструкція xsl: value-of***

Елемент xsl: value-of обчислює вираз, записане в його обов'язковому атрібут select, і перетворює його в рядок. Наприклад, вище ми визначили ім'я об'єкта var2. Щоб отримати значення об'єкта var2, треба записати: `<xsl: value-of select = "$ var2" />`

Якщо в результаті обчислення виразу виходить послідовність, то процесор XSLT версії 1.0 вибере з неї тільки перший елемент, преобразованний в рядок.

***Інструкції управління xsl: if, xsl: for-each, xsl: choose***

Елемент xsl: if запускає конструктор послідовності, що міститься в його тілі, тільки якщо істинно вираз, записане в обов'язковому і єдиному атрибуті test:

```
<Xsl: if test="$x>$y">
<Xsl:   value-of   select="$x"/>більше<xsl:value-ofselect="$y"/>
</xsl: if>
```

У елемента xsl: for-each в обов'язковому і єдиному атрибуті select запісивается вираз, що дає в результаті послідовність. Для кожного члена цієї послідовності виконується конструктор, що міститься в тілі елемента xsl: for-each. В результаті виходить цикл, що виконується стільки разів, скільки елементів у послідовності, отриманої в результаті обчислення вираженія атрибута select.

У елемента xsl: choose немає жодного атрибута, але в його тілі записується один або кілька елементів xsl: when і один необов'язковий елемент xsl: otherwise: <xsl: choose>

```
<xsl when
test="$day=l">Понедельник</xsl:when>
<xsl when
test="$day=2">Вторник</xsl:when> <xsl
when test="$day=3">Среда</xsl:when> <xsl
when test="$day=4">Четверг</xsl:when>
<xsl when
test="$day=5">Пятница</xsl:when>
<xsl when test="$day=6">Суббота</xsl:when>
<xsl when test="$day=7">Воскресенье</xsl:when>
```

```
        <xsl otherwise>Ошибка определения дня недели</xsl:otherwise>
    </xsl:choose>
```

У елемента xsl: when є тільки один обов'язковий параметр test, що містить логічне вираження, і тіло, що містить конструктор послідовності.

Елемент xsl: otherwise не має атрибутів, у нього є тільки тіло, що містить конструктор послідовності.

В інструкції xsl: choose завжди виконується не більш одного варіанту, Варіанти xsl: when проглядаються в порядку їх написання в інструкції xsl: choose. Як тільки буде знайдений варіант з істинним значенням виразу test, він буде виконаний, і результат цього варіанту буде результатом всієї інструкції. Варіанти, які йдуть за ним по порядку, не розглядаються. Якщо жоден варіант не підійде, то результатом інструкції буде результат конструктора, записаного в елементі xsl: otherwise. Якщо елемента xsl: otherwise в інструкції немає, то результатом буде порожній рядок.

***Інструкція xsl: apply-templates***

Елемент xsl: apply-templates, записується найчастіше всередині елемента xsl: template, в найпростішому вигляді порожній:

```
<Xsl:apply-templates/>
```

Він наказує обробити рекурсивно всі вузли-нащадки вузлів, відібраних батьківським елементом xsl: template.

У елемента xsl: apply-templates є необов'язкові атрибути select і mode.

Атрибут select, значенням якого має бути вираз, що дає послідовність вузлів, обмежує обробку тільки зазначеними в ньому вузлами. Атрибут mode вибирає режим обробки з режимів, вже визначених в елементах xsl: template. Режим – це будь-яке ім'я типу QName, але два режими предопределени. Це поточний режим, що відзначається словом #current, і режим за замовчуванням, що приймається при відсутності атрибута mode, або що відзначається явно словом #default.

Вмістом елемента xsl: apply-templates можуть служити елементи xsl: sort і xsl: with-param.

## ПРИКЛАД ВИКОНАННЯ РОБОТИ

Розглянемо використання XSL-таблиць стилів на прикладі. Існують два основні кроки для відображення XML-документа при використанні XSL-таблиці стилів:

1. Створення файлу XSL-таблиці стилів. XSL є додатком XML, тому XSL-таблиця являє собою коректно сформований XML-документ, який відповідає правилам XSL. Подібно будь-якій XML-документу, XSL-таблиця стилів містить простий текст, і можна створити її за допомогою будь-якого текстового редактора.

2. Зв'язування XSL-таблиці стилів з XML-документом. Можливо зв'язати XSL-таблицю стилів з XML-документом, включивши в документ інструкцію з обробки xml-stylesheet, яка має наступну узагальнену форму запису:

```
<? Xml-stylesheet type="text/xsl" href=XSLFilePath?>
```

Тут XSLFilePath – укладений в лапки URL, який вказує місцезнаходження файлу таблиці стилів. Можна використовувати повний URL, наприклад:

```
<?Xml-stylesheet type="text/xsl"href="http:/www.my_site.com/Tab1.xsl"?>
```

Найчастіше використовують неповний URL, який задає місцезнаходження щодо місця розташування XML-документа, що містить інструкцію з обробки xml-stylesheet, наприклад:

```
<? Xml-stylesheet type="text/xsl" href="1.xsl"?>
```

Відносний URL зустрічається частіше, оскільки зазвичай файл таблиці стилів зберігається в тій же папці, де зберігається XML-документ, або в одній з вкладених в неї папок. Можна пов'язати XSL-таблицю стилів з використанням повного URL, таблиця стилів при цьому повинна розміщуватися на тому ж домені, що і XML-документ, з яким її пов'язують.

Наприклад, якщо домен http://docs.soft.com/ містить XML-документ, то і XSL-таблиця стилів повинна розміщуватися на тому ж домені. Зазвичай інструкція по обробці xml-stylesheet додається в пролог XML-документа слідом за оголошенням XML.

Якщо XSL-таблиця стилів пов'язана з XML-документом, то можна відкрити цей документ безпосередньо в Internet Explorer або в іншому браузері, і браузер відобразить XML-документ з використанням інструкцій по перетворенню, що містяться в таблиці стилів. На відміну від таблиць каскадних стилів, якщо з XML-документом зв'язується більше однієї XSL-таблиці стилів, браузер використовує першу таблицю і ігнорує всі інші. Якщо з XML-документом пов'язані і CSS-таблиця і XSL-таблиця стилів, браузер використовує тільки XSL-таблицю стилів.

Якщо XML-документ не пов'язаний ні з CSS-таблицею, ні з XSL-таблицею стилів, браузер відобразить документ за допомогою вбудованої XSL-таблиці, яка використовується за замовчуванням. Ця таблиця стилів відображає вихідний XML-текст у вигляді дерева з можливістю згортання / розгортання рівнів.

***Використання одного шаблону XSL***

На відміну від CSS, що містить правила, XSL-таблиця стилів включає один або кілька шаблонів, кожен з яких містить інформацію для відображення в певній галузі елементів в XML-документі.

*Завдання1:* Створіть xml-документ в будь-якому текстовому редакторі, зберігши його з розширенням xml, попередньо обравши тип файлу – Всі файли або xml, з *інформацією про книгу: назва книги, прізвище, ім'я автора книги, тип обкладинки, кількість сторінок і ціна,* збережіть його, наприклад, книга.xml:

Приклад 1 файлу книга.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
- <книга>
    <название>Web-программирование на Java и JavaScript</название>
  - <автор>
      <имя>Андрей</имя>
      <фамилия>Гарнаев</фамилия>
    </автор>
    <обложка>мягкая</обложка>
    <страницы>1040</страницы>
    <цена>413 рублей</цена>
  </книга>
```

Відкрийте створений файл будь-яким браузером і перевірте коректність відображення даних.

*Завдання 2:* Створіть для xml-документа книга.xml стильовий файл в будь-якому текстовому редакторі з відображенням всієї інформації xml-документа і підписами. Для підписів застосуєте курсив (за допомогою тега span і його атрибута style: <SPAN STYLE = "font-style: italic">), наприклад,

```
<?xml version="1.0" encoding="windows-1251" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  - <xsl:template match="/">
      <H2>Описание книги</H2>
      <SPAN STYLE="font-style:italic">Автор:</SPAN>
      <xsl:value-of select="книга/автор" />
      <BR />
      <SPAN STYLE="font-style:italic">Название:</SPAN>
      <xsl:value-of select="книга/название" />
      <BR />
      <SPAN STYLE="font-style:italic">Цена:</SPAN>
      <xsl:value-of select="книга/цена" />
      <BR />
      <SPAN STYLE="font-style:italic">Тип обложки:</SPAN>
      <xsl:value-of select="книга/обложка" />
      <BR />
      <SPAN STYLE="font-style:italic">Количество страниц:</SPAN>
      <xsl:value-of select="книга/страницы" />
    </xsl:template>
</xsl:stylesheet>
```

Зверніть увагу, що в другому тезі <xsl: stylesheet version = "1.0" xmlns: xsl = "http://www.w3.org/1999/XSL/Transform"> також необхідно вказати атрибут version = "1.0", без нього стильовий файл буде проглядатися без помилок, але при підвантаженні стильового файлу в xml-документ буде видаватися помилка про відсутність даного атрибута.

У тегах <xsl: value-of select = "..." /> вказано «повний шлях» до дочірніх елементів, звернення починається з кореневого елемента «книга», через «/» вказується дочірній елемент, наприклад,

автор: `<xsl:value-of select="книга/автор"/>`.

Для підключення стильового файлу додайте в xml-документ другим тегом `<?xml-stylesheet type="text/xsl" href="книга.xsl"?>`. Відображення XML-документа в Internet Explorer відповідно до інструкцій з таблиці стилів має виглядати наступним чином:

**Описание книги**

*Автор:* Андрей Гарнаев
*Название:* Web-программирование на Java и JavaScript
*Цена:* 413 рублей
*Тип обложки:* мягкая
*Количество страниц:* 1040

Кожна XSL-таблиця стилів повинна мати елемент Документ, представлений нижче (елемент Документ або кореневий елемент - XML-елемент верхнього рівня, який містить всі інші елементи):

```
<Xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <! - один або кілька елементів шаблону  ->
</ Xsl: stylesheet>
```

Елемент Документ xsl: stylesheet служить не тільки сховищем інших елементів, але також ідентифікує документ як XSL-таблицю стилів. Цей елемент є одним з XSL-елементів спеціального призначення, використовуваних в таблиці стилів. Всі XSL-елементи належать простору імен xsl - т. е. ім'я кожного XSL-елемента починається з префікса xsl :, що позначає простір імен. Це простір імен визначається в початковому тегу елемента xsl: stylesheet, наприклад, наступним чином:

```
xmlns: xsl = "http://www.w3.org/1999/XSL/Transform"
```

Це визначення дозволяє використовувати простір імен всередині елементів таблиці стилів.

Елемент Документ xsl: stylesheet XSL-таблиці стилів повинен містити один або кілька шаблонів елементів. Елемент Документ з прикладу 2 містить тільки один шаблон, який має наступну форму:

```
<Xsl: template match = "/">
 <! - дочірні елементи ... ->
</ Xsl: template>
```

Браузер використовує шаблон для відображення певної гілки елементів в ієрархії XML-документа, з яким пов'язана таблиця стилів. Атрибут match шаблону вказує на певну галузь (атрибут match аналогічний селектору в правилі CSS). Значення атрибута match носить назва зразка (pattern). Зразок в даному прикладі ( "/") являє кореневий елемент всього XML-документа. Цей шаблон, таким чином, містить інструкції для відображення всього XML-документа. Кожна XSL-таблиця стилів повинна містити один і тільки один шаблон з атрибутом match, який має значення "/". також можна включити один або кілька додаткових шаблонів з інструкціями для відображення певних підлеглих гілок в структурі XML-документа; кожна з них повинна мати зразок, який відповідає певній гілки.

Кореневої зразок ( "/") не представляє елемент Документ (або кореневий елемент) XML-документа. Він представляє весь документ, для якого елемент Документ є дочірнім.

Приклад повного опису шаблону з даної таблиці стилів:

```
<Xsl: template match = "/">
 <H2> Опис книги </ H2>
 <SPAN STYLE = "font-style: italic"> Автор: </SPAN>
```

```
 <Xsl: value-of select = "книга / автор" /><BR/>
 <SPAN STYLE = "font-style: italic"> Назва: </SPAN>
 <Xsl: value-of select = "книга / назва" /><BR/>
 <SPAN STYLE = "font-style: italic"> Ціна: </SPAN>
 <Xsl: value-of select = "книга / ціна" /><BR/>
 <SPAN STYLE = "font-style: italic"> Тип обкладинки: </SPAN>
 <Xsl: value-of select = "книга / обкладинка"/> <BR/>
 <SPAN STYLE="font-style: italic">Кількість сторінок:</SPAN>
 <Xsl: value-of select = "книга / сторінки" />
</ Xsl: template>
```

Шаблон містить два види XML-елементів:

1. XML-елементи, що представляють HTML-розмітку. Прикладами подібного виду XML-елемента з даної таблиці стилів є:

<H2> Опис книги </ H2> який відображає заголовок другого рівня, <SPAN STYLE = "font-style: italic"> Автор: </ SPAN> який відображає блок тексту, набраного курсивом (Автор :), і <BR/> який створює порожній рядок.

Всі ці XML-елементи є коректно сформованими і представляють стандартні HTML-елементи. Браузер просто копіює кожен HTML-елемент безпосередньо на вихід HTML, який сприймає і відображає їх. Кожен з елементів, що представляють HTML-розмітку, повинен бути коректно сформованим XML-елементом, а також стандартним HTML-елементом (XSL-таблиця стилів є XML-документом). отже, не можна використовувати HTML-конструкції, які не є коректно сформованим XML, такі, як елементи, що складаються тільки з початкового тега.

2. XSL-елементи. Прикладами XSL-елементів з даної таблиці стилів є елементи xsl: value-of, наприклад:

```
<Xsl: value-of select="книга/автор"/>
```

Браузер відрізняє XSL-елемент від елемента, що представляє HTML, оскільки перший має в якості префікса опис простору імен `xsl:`. XSL-елементи в шаблоні чи не копіюються на вихід HTML, вони лише містять інструкції по вибору і модифікації даних XML, або використовуються для виконання інших завдань. XSL-елемент value-of додає текстовий вміст певного XML-елемента, а також будь-яких його дочірніх елементів, які він має, в вихідний модуль HTML, який сприймається і відображається браузером. Вказується певний XML-елемент завданням зразка, який присвоюється атрибуту select XSL-елемента value-of. В розглянутому вище прикладі елементу value-of атрибуту select присвоєно зразок "книга / автор", що призводить до висновку текстового вмісту елемента автор XML-документа. Текстовий вміст елемента автор складається з символьних даних, що належать двом його дочірнім елементам, прізвище та ім'я.

Зверніть увагу, що XML-елемент у зразку задається за допомогою оператора шляху (в даному випадку книга / автор), який визначає місцезнаходження елемента в ієрархії XML-документа. (Оператор шляху аналогічний шляху до файлу, який операційна система використовує для вказівки місцезнаходження файлу або папки).

Головний момент, на який тут слід звернути увагу, полягає в тому, що оператор шляху в значенні атрибута select відноситься до поточного елементу. Кожен контекст всередині XSL-таблиці стилів відноситься до поточному елементу. Оскільки розглянутий приклад шаблону відноситься до кореневого елементу всього документа (за допомогою установки атрибута match = "/"), поточним «елементом» для даного шаблону є кореневої елемент документа. (В даному випадку поточний елемент не володіє відповідним літералом, а є батьком елемента Документ).

Таким чином, всередині цього шаблону оператор шляху книга / автор вказує на елемент автор, вкладений в елемент книга, вкладений в кореневій елемент документа. (Оператор шляху в значенні атрибута select аналогічний неповного шляху до файлу, що задає місцезнаходження файлу щодо поточної робочої папки). Якщо атрибут select для XSL-елемента value-of відсутня, елемент буде здійснювати вивід текстового вмісту плюс текстове вміст всіх дочірніх елементів поточного елемента. (У нашому прикладі поточним є кореневий елемент, пропуск атрибута select приведе до виведення всіх символьних даних XML-документа).

Метою представленого в розглянутому прикладі шаблону елементів є відображення тексту назви для кожного з дочірніх XML-елементів в документі (автор, назва, ціна, обкладинка і сторінки) плюс текстового вмісту кожного елемента. Зверніть увагу, що порядок елементів value-of в шаблоні визначає порядок, в якому браузер відображає ці елементи. З цієї простої таблиці стилів видно, що XSL-таблиця стилів є набагато більш гнучкою, ніж CSS, яка завжди відображає елементи в тому порядку, в якому вони слідують у документі.

XSL-таблиця стилів повідомляє браузеру, як відобразити XML-документ шляхом виборчого перетворення XML-елементів в блок HTML-розмітки, який сприймається і відображається браузером аналогічно розмітці, що міститься на HTML-сторінці. Не потрібно включати в XSL-шаблон елементи, що представляють елементи HTML або body, які є стандартними складовими частинами HTML-сторінки, оскільки браузер сам ефективно їх формує.

В випадку, якщо документ містить кілька елементів книга, методика, використана в прикладі 2, здатна відобразити тільки один з елементів. Наприклад, XML-документ містить наступний елемент Документ:

```
<?xml version="1.0" encoding="windows-1251" ?>
- <Список_книг>
  - <книга>
      <название>Введение в SQL</название>
    - <автор>
        <имя>Мартин</имя>
        <фамилия>Грубер</фамилия>
      </автор>
      <обложка>твердая</обложка>
      <страницы>378</страницы>
      <цена>618 рубля</цена>
    </книга>
  - <книга>
      <название>Microsoft SQL Server 2012. Основы T-SQL</название>
    - <автор>
        <имя>Ицик</имя>
        <фамилия>Бен-Ган</фамилия>
      </автор>
      <обложка>мягкая</обложка>
      <страницы>400</страницы>
      <цена>743 рубля</цена>
    </книга>
  - <книга>
      <название>SQL. Справочник</название>
    - <автор>
        <имя>Кевин Е.</имя>
        <фамилия>Кляйн</фамилия>
      </автор>
      <обложка>мягкая</обложка>
      <страницы>656</страницы>
      <цена>1259 рублей</цена>
    </книга>
</Список_книг>
```

Припустимо, що таблиця стилів, яка використовується для відображення основних напрямів, містить наступний шаблон: Наприклад 4. книги.xsl

```
<?xml version="1.0" encoding="windows-1251" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/ Transform">
  - <xsl:template match="/">
      <H2>Описание книг</H2>
      <SPAN STYLE="font-style:italic">Автор:</SPAN>
      <xsl:value-of select="Список_книг/книга/автор" />
      <BR />
      <SPAN STYLE="font-style:italic">Название:</SPAN>
      <xsl:value-of select="Список_книг/книга/название" />
      <BR />
      <SPAN STYLE="font-style:italic">Цена:</SPAN>
      <xsl:value-of select="Список_книг/книга/цена" />
      <BR />
      <SPAN STYLE="font-style:italic">Обложка:</SPAN>
      <xsl:value-of select="Список_книг/книга/обложка" />
      <BR />
      <SPAN STYLE="font-style:italic">Количество страниц:</SPAN>
      <xsl:value-of select="Список_книг/книга/страницы" />
    </xsl:template>
  </xsl:stylesheet>
```

Цей шаблон використовує методику, описану в прикладі 2. Зверніть увагу, що зразок привласнюються кожному атрибуту select починається з вказівки елемента Документ, в даному випадку Спісок_кніг (наприклад, "Спісок_кніг/книга/автор"). Кожен зразок, однак, відповідає трьом різним елементам. Наприклад, "Спісок_кніг/книга/автор" відповідає елементу автор для всіх трьох елементів книга. У подібній ситуації браузер використовує тільки перший з відповідних елементів. Щоб відобразити всі відповідальні зразком елементи, слід використовувати XSL-елемент for-each, який викликає повторний висновок для кожного з містяться в XML-файлі елементів. XSL-таблиця стилів, демонструє дану методику, ця таблиця стилів пов'язана з XML-документом. Приклад 5. Модифікований стильовий файл

```
<?xml version="1.0" encoding="windows-1251" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/ Transform">
  - <xsl:template match="/">
      <H2>Описание книг</H2>
    - <xsl:for-each select="Список_книг/книга">
        <SPAN STYLE="font-style:italic">Название:</SPAN>
        <xsl:value-of select="название" />
        <BR />
        <SPAN STYLE="font-style:italic">Автор:</SPAN>
        <xsl:value-of select="автор" />
        <BR />
        <SPAN STYLE="font-style:italic">Обложка:</SPAN>
        <xsl:value-of select="обложка" />
        <BR />
        <SPAN STYLE="font-style:italic">Количество страниц:</SPAN>
        <xsl:value-of select="страницы" />
        <BR />
        <SPAN STYLE="font-style:italic">Цена:</SPAN>
        <xsl:value-of select="цена" />
        <P />
      </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

## ЗАВДАННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Написати таблицю стилів на XSL для відображення XML-документа з лабораторной роботи №1. Створіть XSL-файл на основі схеми (див. завдання до лабораторної роботи №1). У звіті до лабораторної роботи необхідно надати лістинг вашого XSL-файлу і результат відображення вашого XML-файлу різними браузерами.

## КОНТРОЛЬНІ ПИТАННЯ

1. Основні конструкції мови XSL
2. Візуалізація документа XML
3. Простір імен. Визначення. Використання.
4. Простір імен. Використання і декларація простору імен.
5. Зв'язування XSL-таблиці стилів з XML-документом
6. Переваги та недоліки использование одного шаблоном XSL

## ЛАБОРАТОРНА РОБОТА №5

**Тема:** відображення XML-документа за допомогою об'єктної моделі документа DOM – Document Object Model.

**Мета роботи**: Вивчити основи бібліотеки об'єктної моделі документа DOM. Навчитися відображати XML- документи з використанням DOM.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

При написанні сценарію ви створюєте HTML-сторінку, пов'язуєте її з XML-документом і маєте доступ до індивідуальних XML-елементів за допомогою спеціально написаного коду сценарію (JavaScript або Microsoft Visual Basic Scripting Edition [VBScript]). Браузер сприймає XML-документ як об'єктну модель документа (Document Object Model – DOM), що складається з великого набору об'єктів, властивостей і команд. Написаний код дозволяє здійснювати доступ, відображення і маніпулювання XML-елементами.

У браузерах Internet Explorer знаходяться вбудовані бібліотеки DOM. Для сценаріїв на стороні клієнта є безліч об'єктів для роботи з XML-документом, найважливіші з них, об'єкти `XMLDOMDocument`, `XMLDOMNode`, `XMLDOMNodeList`, `XMLDOMParseError`, що представляють інтерфейс для доступу до всього документа, окремим його вузлів і піддерев, що надають необхідну для налагодження інформацію про що відбулися помилки аналізатора відповідно.

Об'єкт XMLDOMNode, який реалізує базовий DOM інтерфейс Node, призначений для маніпулювання з окремим вузлом дерева документа. Його властивості та методи дозволяють отримувати і змінювати повну інформацію про поточні вузли – його тип (dataType, nodeType, nodeTypeString), назва (baseName, prefix, nodeName), його вміст (attributes, text, nodeValue, childNodes) і т.д .

*При виконанні даної лабораторної роботи можуть бути корисні наступні властивості:*

nodeName – Повертає повну назву (разом з Namespace атрибутом) поточного вузла у вигляді рядка. Доступно тільки для читання.

baseName – Повертає назву елемента без префікса Namespace. Тільки для читання.

prefix – Повертає Namespace префікс. Тільки для читання.

dataType – Визначає тип вмісту поточного вузла (описуваний схемами даних). Доступно для запису і читання.

nodeType – Повертає тип поточного вузла. Тільки для читання.

nodeTypeString – Повертає тип вузла у вигляді тексту. Тільки для читання.

attributes – Повертає список атрибутів поточного вузла у вигляді колекції XMLDOMNamedNodeMap. Якщо атрибутів немає, то властивість length буде мати нульове значення. Для тих вузлів, у яких не може бути атрибутів, повертається null. Доступно тільки для читання.

nodeValue – Повертає вміст поточного вузла. Доступно для читання і запису.

childNodes – Для тих вузлів, які мають дочірні елементи, повертає їх список у вигляді XMLDOMNodeList. У тому випадку, якщо дочірніх елементів немає, значення властивості length списку дорівнює нулю. Тільки для читання.

lastChild – Повертає останній дочірній елемент або null, якщо таких немає. Властивість доступна тільки для читання.

firstChild – Повертає перший дочірній елемент або null. Тільки для читання.

nextSibling – Повертає наступний дочірній елемент. Тільки для читання.

previousSibling – Повертає попередній дочірній елемент. Доступно тільки для читання.

parentNode – Містить посилання на батьківський елемент. У тому випадку, коли такого елемента немає, повертає null. Доступно тільки для читання.

## ВИКОНАННЯ РОБОТИ

Об'єкт XMLDOMDocument представляє верхній рівень об'єктної ієрархії і містить методи для роботи з документом:

- його завантаження (readyState, load (url), loadXML (xmlString),
- save (objTarget), abort () і т.д.),
- створення в ньому елементів ( createElement (tagName)),
- атрибутів (createAttribute (name)),
- коментарів (createComment (data)) і т.д.

Багато властивостей і методи цього об'єкту реалізовані також в розглянутому вище класі Node, тому що документ може бути розглянутий як кореневий вузол з вкладеними в нього піддеревами.

Об'єкт XMLDOMNodeList є список вузлів – піддерева містять методи, за допомогою яких можна організувати процедуру обходу дерева. наприклад:

- length – число елементів списку вузлів;
- item (i) – вибір і-того елемента зі списку. Повертає об'єкт XMLDOMNode;
- nextNode () – вибір наступного елемента в списку, якщо такого елемента немає, то повертає null. Перший виклик цього методу поверне посилання на перший елемент списку;

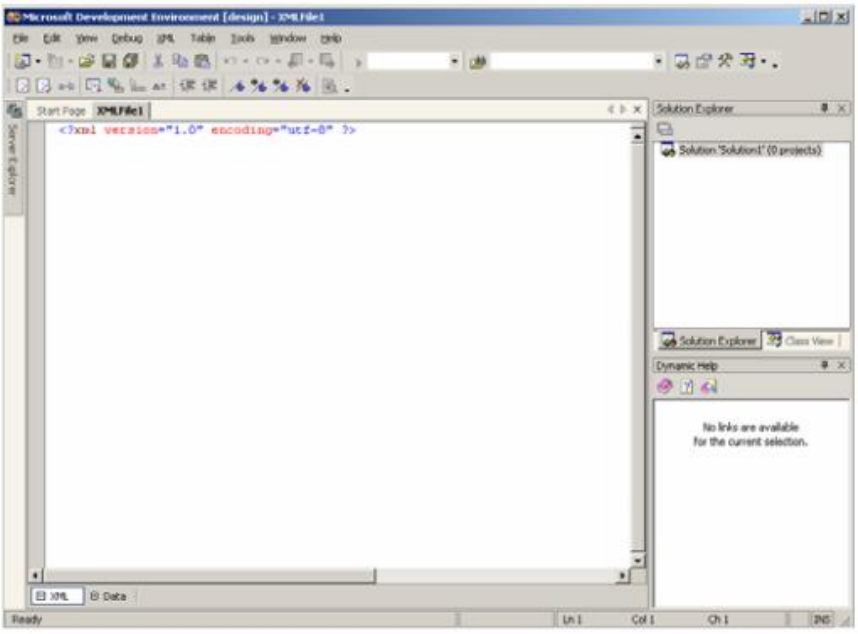
- reset () – скидання внутрішнього покажчика поточного елемента.

Об'єкт XMLDOMParserError дозволяє отримати всю необхідну інформацію про помилку, що сталася в ході розбору документа. Всі властивості цього об'єкта доступні тільки для читання. Основні властивості:

- errorCode – містить код виникла помилки або 0, якщо такої не сталося;
- url – повертає URL оброблюваного документа;
- filepos – повертає зсув щодо початку файлу фрагмента, в якому виявлена помилка;
- line – містить номер рядка, що містить помилку;
- linepos – позицію помилки в рядку, в якій була виявлена помилка;
- srcText – містить повний текст рядка, в якій сталася помилка.

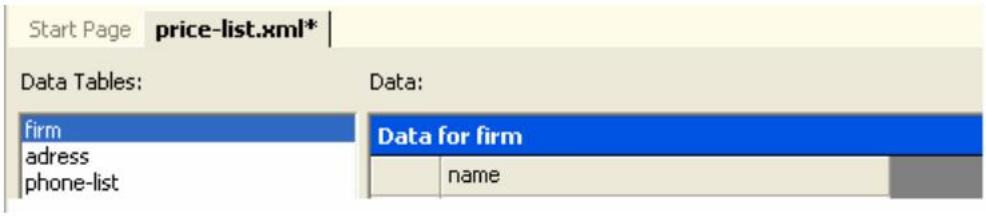
### *Створення XML документа з використанням Microsoft Visual Studio .Net*

Microsoft Visual Studio .Net дозволяє досить легко створювати XML документи і XSD схеми до них. Для того щоб створити XML документ необхідно в середовищі розробки Visual Studio .Net виконати команду меню `File – New – File ...` в діалоговому вікні `New File` в наборі `Templates` виділити значок XML File. В результаті на основному робочому полі з'явиться наступного виду сторінка:




На даній сторінці можна набрати XML документ. Причому при написанні відкривається тега закривається тег буде з'являтися автоматично. Заповнити XML

документ даними можна за допомогою розділу Data. Для цього в низу вікна потрібно знайти кнопку "Data". В результаті сторінка придбає приблизно такий вигляд:




Введені таким чином дані будуть додані в XML документ. Для того щоб для створеного XML документа написати XSD схему необхідно на робочому полі натиснути правою кнопкою миші і в випадаючому списку вибрати Create Schema.

## ЗАВДАННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Створіть HTML-сторінку та зв’яжіть її з XML-документом (з лабораторної роботи №1). Напишіть код сценарію, використовуючи JavaScript для доступу к індивідуальним XML-документам.

У звіті до лабораторної роботи необхідно надати лістинг вашого XSL-файлу і скриптів, а також демонстрацію роботи скриптів.

## КОНТРОЛЬНІ ПИТАННЯ

1) Об'єктна модель документа: переваги застосування.

2) Навіщо потрібна модель DOM.

3) Використання моделі DOM на сервері.

4) Використання моделі DOM у клієнта.

5) Який метод використовують для створення нового елемента в DOM?

6) Як реалізує базовий DOM інтерфейс Node

7) Який об’єкт являє верхній рівень об'єктної ієрархії і містить методи для роботи з документом: його завантаження, аналізу, створення в ньому елементів, атрибутів, коментарів?

## ЛАБОРАТОРНА РОБОТА №6

**Тема:** Схема XSD. відображення XML-документа за допомогою стандарту XSD (XML Schema Definition).

**Мета роботи**: Вивчити основні елементи та простір імен XML Schema для створення XSD схеми документів.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

На зміну DTD прийшов стандарт консорціуму W3C – XML Schema, званий також XSD (XML Schema Definition).

Кореневим елементом в схемі XML є елемент Schema, який містить всі інші елементи в документі схеми. В рамках кореневого елемента схеми XSD атрибутом xmlns визначається простір імен XML Schema, яке містить елементи і атрибути XSD схеми.

```
<Xsd: schema xmlns: xsd = "http://www.w3.org/2001/XMLSchema">
```

Всі елементи XSD починаються із префікса xsd:, який вказується для простору імен XSD, оголошеного в кореневому елементі примірника схеми.

У самому XML-документі, який перевіряється за допомогою схеми, має міститися оголошення простору імен. ***Простір імен*** – це іменована сукупність імен елементів і атрибутів, що служить для забезпечення їх унікальності в XML-документі. Простір імен завжди вказується в кореневому елементі примірника документа з допомогою атрибута xmlns:

```
xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance»
```

У схемах XSD елементи, які використовуються в документах XML, бувають двох типів: *прості (simpleType)* і *складні (complexType)*. Елементи складних типів можуть містити атрибути, а також дочірні елементи. Елементи простих типів такими можливостями не володіють.

| Простий тип | Опис |
|---|---|
| string | буквено-цифровий рядок |
| integer | …-1,0,1,… |
| time | 13:20:00.000, 13:20:00.000-05:00 |
| date | 1999-05-31 |
| gYear | 1999 |
| anyURL | http://www.example.com/, http://www.example.com/doc.html#ID5 |

***Елементи XSD-схем:***

| **Елемент** | **Опис** |
|---|---|
| annotation | Родительский элемент элементов-комментариев <appInfo> и <documentation> |
| appInfo | Элемент-комментарий. Задает титул схемы |
| attribute | Атрибут |
| choice | Выбор других элементов. Аналог оператора "\|" в DTD |
| complexContent | Ограничения или расширения модели содержимого сложного типа |
| complexType | Элемент сложного типа |
| documentation | Элемент-комментарий. Предоставляет информацию о схеме |
| element | Элемент |
| restriction | Ограничение элемента |
| schema | Корневой элемент схемы |
| sequence | Последовательность других элементов. Анналог оператора "," в DTD |
| simpleContent | Модель, содержимое которой представляет только символьные данные |
| simpleType | Элемент простого типа |

***Обмеження входжень в схемах XSD***

На відміну від інших мов опису схем, XSD дозволяє вам визначити кількість входжень елемента з визначеної точністю. Можливо задати мінімальне і максимальне кількість входжень елемента за допомогою атрибутів minOccur і maxOccur елемента xsd: element відповідно. На можливі значення цих атрибутів накладаються определен- ні обмеження:

| | |
|---|---|
| minOccur=”0” | вхождение элемента необязательно |
| minOccur=”1” | предусмотрено одно вхождение элемента |
| maxOccur=”1” | |
| minOccur и maxOccur не указаны | |
| minOccur=”2 (3,4,…)” | минимальное число вхождений равно 2 (3,4,..) |
| maxOccur=”2 (3,4,…)” | максимальное число вхождений равно 2 (3,4,..) |
| maxOccur=”unbounded” | число вхождений не ограничено |

***Атрибути в схемах XSD***

Оголошення атрибутів XML документів в схемах XSD дуже схоже на оголошення елементів, за винятком того, що атрибути оголошуються за допомогою оголошень attributes, а НЕ element.

***Атрибути-обмеження XSD-схем:***

| **Атрибут** | **Опис** |
|---|---|
| length | Длина |
| maxLength | Максимальная длина |
| maxInclusive | Максимальное значение включительно |
| minExclusive | Минимальное значение |

**Інші атрибути:**

| **Атрибут** | **Опис** |
|---|---|
| fixed | Фиксированное значение элемента или атрибута |
| name | Название элемента или атрибута |
| ref | Задание ссылки на глобально определенный элемент |
| schemaLocation | Определение местоположения схемы |
| type | Тип элемента |
| use | Є елемент обов'язковим чи ні. Можливі значення атрибута use:<br>– required – атрибут є обов'язковим і може мати будь-який знання;<br>– optional – атрибут є необов'язковим і може мати будь-яке значення;<br>– fixed – значення атрибута фіксоване, і його можна встановити з допомогою на гою атрибута value;<br>– default – якщо атрибута немає, його значення дорівнює значенню за замовчуванням, встановленому для атрибута value (якщо атрибут присутній, то його значення дорівнює значенню, яке присвоюється йому в цьому документі);<br>– prohibited – атрибут не повинен відображатися. |
| value | Значение элемента схемы |
| xsi:schemaLocation | Реальное местоположение элемента в XML-документе |
| xsi:type | Реальный тип элемента в XML-документе |

## ПРИКЛАД ВИКОНАННЯ РОБОТИ

*Документ:*

```
<?xml version="1.0" encoding="UTF-8"?>
<note
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="3.xsd">
<message number="10" date="2009-10-29" from="Ольга Николаевна">
Не забудь купить хлеб по дороге с работы домой
</message>
</note>
```

*Схема XSD(3.xsd):*

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="messageType">
<xsd:simpleContent>
<xsd:restriction base="xsd:string">
```

```
  <xsd:attribute name="number"   type="xsd:integer"
use="required"/>
  <xsd:attribute name="date" type="xsd:date" use="required"/>
  <xsd:attribute name="from" type="xsd:string" use="required"/>
  </xsd:restriction>
  </xsd:simpleContent>
  </xsd:complexType>
  <xsd:element name="note">
  <xsd:complexType>
  <xsd:sequence>
  <xsd:element name="message" type="messageType"/>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
```

Аналіз

Елемент simpleContent і елемент complexType не містять атрибут name. Можливо вказати ці атрибути, якщо плануєте повторно використовувати виділення змісту в інших місцях схеми. Наприклад, елемент complexType називається messageType. Оголошення елемента message містить посилання (type = "messageType") на об'єкт явище складного типу з оголошенням атрибута. С допомогою дескриптора <xsd: restriction base = "xsd: string"> оголошується обмеження типу даних для атрибута. Кожен атрибут відноситься до типу string, тому в подальшому він обмежується оголошенням типів даних (integer, date, string), а також атрибутом use (кожен атрибут обов'язковий в перевіряється документі XML).

Елемент sequence включений тільки для того, щоб показати типове розташування і синтаксис, використовувані в екземплярі, що містять кілька дочірніх елементів, для чого необхідні обмеження послідовностей.

## ЗАВДАННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Напишіть для XML-документу схему XSD. 1. Накладіть на цей документ ряд обмежень:

- в документі допускається не більше двох елементів notes;
- входження елементів notes необов'язково;
- елемент number повинен передувати елементу message;
- повинен існувати як мінімум один елемент message.

У звіті до лабораторної роботи необхідно надати XSD-схему XML-документу згідно з варіантом завдання.

## КОНТРОЛЬНІ ПИТАННЯ

1) Для чого потрібна XSD схема?

2) Які типи елементів в XSD схемах ви знаєте? У чому їх відмінність?

3) Як задаються атрибути XML-документа в XSD схемах?

4) Як за допомогою XSD схеми задати певну послідовність елементів в XML документі?

5) Простір імен. Правила оголошення простору імен.

6) Обмеження входжень в схемах XSD.

## ЛАБОРАТОРНА РОБОТА №7

**Тема:** Використання PATH-висловів мови запитів XQUERY

**Мета роботи**: Навчитися писати запити на мові XQuery до XML-даних, використовуючи PATH-вирази.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

Path-вирази використовуються для навігації по вхідному документу для вибірки елементів і їх атрибутів. Вони складаються з однієї або декількох частин розділених / або //. наприклад:

`doc ("catalog.xml")/catalog/product` – вибірка всіх product-нащадків з файлу `doc ("catalog.xml")/catalog` – повертає весь елемент catalog doc `("catalog.xml")//product` – повертає всі елементи product, які можуть розташовуватися в документі де-завгодно

`doc ("catalog.xml")//product/@dept` – повертає атрибути dept у відповідних елементів product

`doc ("catalog.xml")/catalog/*` – повертає всіх нащадків елемента `catalog doc ("catalog.xml")/catalog/*/number` – повертає всі елементи number, які є нащадками елемента catalog

Path-вирази повертають вузли в порядку їх слідування в документі. Предикати використовуються в path-виразах, щоб фільтрувати результати за спеціальним критерієм. наприклад:

`product [name="FloppySunHat"]-все product зі значенням дитини name рівним 'Floppy Sun Hat'`

`product [number <500]` – все product зі значенням дитини number менше `500 product [@dept="ACC"]` – все product зі значенням атрибута dept рівними 'ACC'

`product [desc]` – все product, що мають принаймні одну дитину desc `product [@dept]` – все product, що мають атрибут dept

`product [@dept]/number` – всі діти number продуктів, які мають атрибут dept

У предикатів можуть обчислюватися логічні вирази, якщо результат обчислення дорівнює true, то відповідний вузол повертається, якщо false, то немає. Число 0, NaN, рядок нульової довжини, порожня послідовність – візьмуть значення false.

Предикати також можливо використовувати для вказівки порядкової позиції елемента (вузла) в послідовності, ці предикати іноді називаються позиційний предикати, наприклад: `doc ("catalog.xml")/catalog/product`

Будь-який предикатний вираз, який обчислюється як ціле значення, може бути використано як позиційний предикат. Якщо вкажемо номер, який більше, ніж кількість елементів в контекстної послідовності, повернеться порожня послідовність (але не помилка), наприклад:

```
doc ( "catalog.xml")/catalog/product [99]
```

## ПРИКЛАД ВИКОНАННЯ РОБОТИ

Предикати можуть бути пов'язані один з одним, щоб фільтрувати елементи більш, ніж за одним критерієм, наприклад:

```
doc ( "catalog.xml") / catalog / product [number <500] [@ dept = "ACC"]
```

або те ж саме:

```
doc ( "catalog.xml") / catalog / product [number <500 and @dept = "ACC"]
```

`doc("catalog.xml")/catalog/product/name[1]` – звертається до першого name для кожного product;

`Doc("catalog.xml")/catalog/product/name)[1]` – звертається до першого елементу name в документі;

`doc("catalog.xml")/catalog/descendant::name[1]` – те ж саме, що в попередньому випадку

`doc("catalog.xml")/catalog//name[1]` – звертається до першого елементу name в будь-якому тезі (якщо він там є).

Також множинні предикати можна комбінувати з позиційними предикатами, наприклад:

`doc("catalog.xml")/catalog/product[@dept="ACC"][2]- другий product, який має значення атрибута dept рівне "ACC".`

Порядок предикатів значущий, такий вираз має вже трохи інший зміст:

`doc("catalog.xml")/catalog/product[2][@dept="ACC"]` – другий product, якщо він має значення атрибута рівне "ACC".

Предикати можуть містити виклики функцій, наприклад: `doc("catalog.xml")/catalog/product[contains(@dept,"A")]` – поверне все product, атрибут dept яких містить в своєму значенні букву 'A'.

Предикати можуть містити умовні вирази:

```
doc("catalog.xml")/catalog/product[if($descFilter) then desc else true ()]
```

Використання комбінованих послідовностей:

```
doc("catalog.xml")/catalog/product[* except number]
```

Використання основних порівнянь:

```
doc("catalog.xml")/catalog/product[@dept=("ACC","WMN","MEN")]
```

`doc("catalog.xml")/catalog/product[position() mod 3=0]` – повертає кожен третій product з каталогу.

Предикат може містити вбудовані в нього інші предикати: `doc("catalog.xml")/catalog/product[*[3][self::colorChoices]]` – повертає всі product, третя дитина яких colorChoices.

Предикати можуть використовуватися не тільки в path-виразах, наприклад: `(1 to100) [.mod 5=0]` – повертає числа від 1 до 100, які діляться на 5 без залишку.

`(@Price,0.0)[1]` – повертає атрибут price, якщо він існує або нуль в іншому випадку.

Шляхи в запиті не зобов'язані бути статичними, вони можуть обчислюватися динамічно, для цього використовується динамічний PATH.

Наприклад, якщо попередньо визначені змінні elementName, searchValue варто виконати такий запит:

```
doc("catalog.xml")//*[name()=$elementName][.=$ SearchValue]
```

Доступ до XML документу здійснюється за допомогою вбудованої функції doc (), наприклад:

`doc("catalog_n.xml")` – поверне весь XML документ.

## ЗАВДАННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

Використовуйте створений у лабораторній роботі №1 XML-документ, згідно з обраним варіантом завдання, що представляє інформацію по певній предметної області.

Напишіть для XML-документу за допомогою PATH-виразів *п'ять* запитів на вибірку інформації, згідно предметної області, яка надана в варіанті.

У звіті до лабораторної роботи необхідно надати розроблені PATH-вирази та результати вибірки необхідної інформації з XML-документа.

**КОНТРОЛЬНІ ПИТАННЯ**

1) У якому порядку PATH-вирази повертають вузли документа?

2) Які основні осі використовуються в PATH-виразах?

3) Для чого використовується груповий символ?

4) Для чого використовуються предикати в PATH-виразах?

5) Чим відрізняється використання основних операторів порівняння від операторів порівняння за значенням?

6) Що станеться, якщо в позиційному предикаті буде вказано номер, більший, ніж кількість елементів в контекстної послідовності?

7) Що таке контекстний вузол, як він позначається і як він використовується?

# ЛІТЕРАТУРА

**Основна:**


1. Гнатовська Г.А. XML-технології. Конспект лекцій. ОДЕКУ, 2019. – 75 с.
2. Проценко О. Б. Web-програмування та web-дизайн. Технологія XML. Навчальний посібник. – Суми: Видавництво СумДУ, 2009. – 126 с.
3. В.П. Молчанов, О. К. Пандорін. Технології розробки WEB-ресурсів: навчальний посібник. – Харків: ХНЕУ ім. С. Кузнеця, 2019. – 130 с.


**Додаткова:**


4. В.П. Молчанов. Основи проектування WEB-видань: навч. посіб. Харків: ХНЕУ ім. С. Кузнеця, 2017. – 150 с.
5. О.О. Мосіюк. WEB-технології. Частина 1. Житомир: Вид-во ЖДУ ім. Івана Франка, 2020. – 56 с.
6. Степаненко О.О. Програмування Інтернет-застосувань: конспект лекцій для студентів спеціальності «Інженерія програмного забезпечення» усіх форм навчання. Запоріжжя, 2016. – 66 с.
7. О.О.Шумейко. Конспект лекцій з дисципліни «Технології створення Web-застосувань» для здобувачів вищої за ОПП «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення». – Кам’янське: ДДТУ, 2019– 124 с.