

Лабораторна робота №1

Створення простих web-застосунків засобами Java EE (Enterprise Edition)

Мета: придбати практику у створенні простих web-додатків, заснованих на зв'язках JSP – Servlet - JSP та JSP - JSP (JSP-Java Server Page) а потім розгортувати (deploy) їх на сучасних Java web-серверах та web-контейнерах.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал.

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям. Повертатися із сервлета на JSP треба за допомогою об'єкту класу RequestDispatcher.
2. Створити JSP, яка викликає сервлет, який в свою чергу «на льоту» створює сторінку, що містить три блоки будь-якого тексту, котрі відрізняються один від одного за розміром, кольором, типом вирівнювання та формою тексту; кольором, зображенням (із його параметрами) фону; розмірами та відступами між блоками. Під час виконання завдання треба врахувати те, що за номером власного варіанту частина інформації (вміст та css-параметри абзаців) має розташовуватися в діалогових об'єктах (текстові поля, області ,списки, радібатони, чекбокси та ін.) форми, з якої визивається сервлет, а інша частина css-параметрів має бути зчитана сервлетом із відповідної таблиці чи таблиць бази даних (БД). База даних та таблиця(ї) мають бути створені заздалегідь і мати будь-яку структуру та вміст у будь-якій системі управління базами даних.

№ варіанту	Вміст абзаців	Текстові параметри	Фонові параметри	Розміри та відступи
1.	БД	JSP	JSP	БД
2.	JSP	БД	БД	JSP
3.	БД	БД	JSP	JSP
4.	JSP	JSP	БД	БД
5.	БД	JSP	БД	JSP
6.	JSP	БД	JSP	БД
7.	БД	JSP	БД	БД
8.	БД	БД	JSP	БД
9.	БД	БД	БД	JSP
10.	JSP	БД	БД	БД
11.	БД	JSP	JSP	JSP
12.	JSP	JSP	JSP	БД

3. Створити JSP, яка викликає сервлет, який в свою чергу «на льоту» створює сторінку із зображеннями згідно із топологією власного варіанту

№ варіанту	Форма топології
1	2
1.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
2.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
3.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
4.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
5.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>

1	2				
6.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
7.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
8.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
9.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
10.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
11.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
12.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5

Посилання на зображення задаються на JSP, розміри зображень – з відповідної БД

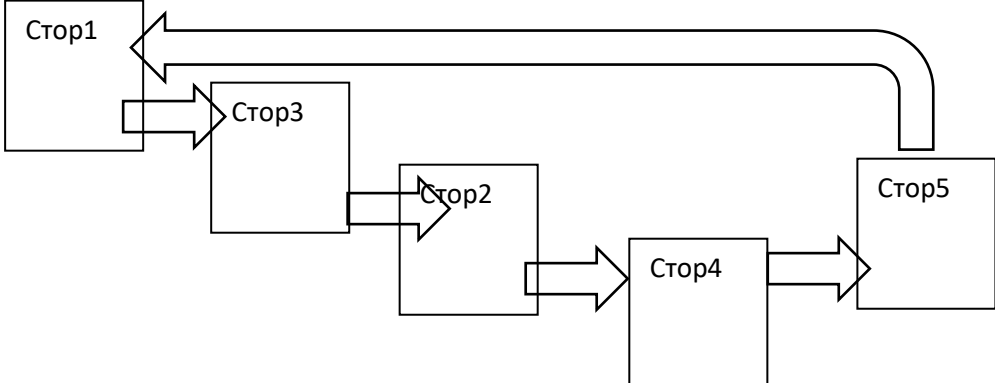
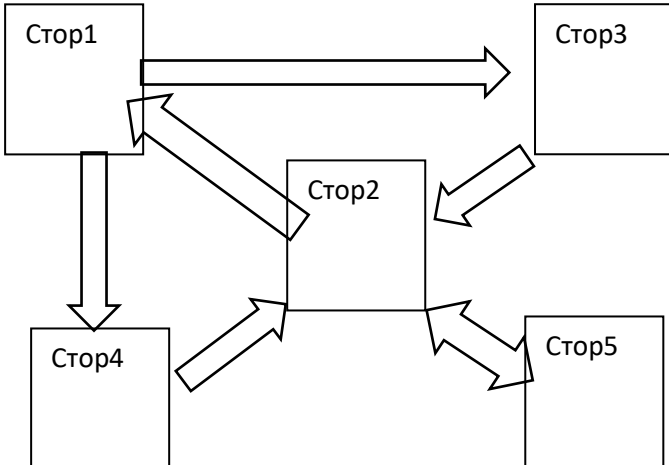
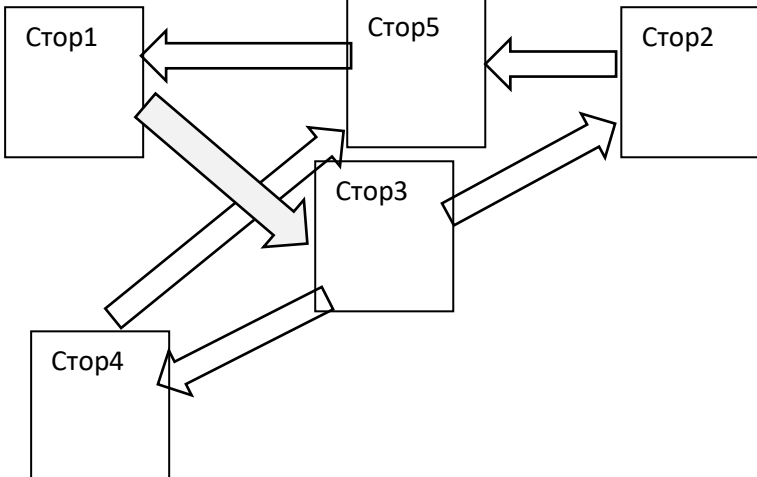
№ варіанту	Форма топології		
1	2		
1.	Зобр1	Зобр2	Зобр3
2.	Зобр1	Зобр2	Зобр3

1	2		
3.	3o6p1	3o6p2	3o6p3
4.	3o6p1	3o6p2	3o6p3
5.	3o6p1	3o6p2	3o6p3
6.	3o6p1	3o6p2	3o6p3
7.	3o6p1	3o6p2	3o6p3
8.	3o6p1	3o6p2	3o6p3
9.	3o6p1	3o6p2	3o6p3
10.	3o6p1	3o6p2	3o6p3
11.	3o6p1	3o6p2	3o6p3
12.	3o6p1	3o6p2	3o6p3

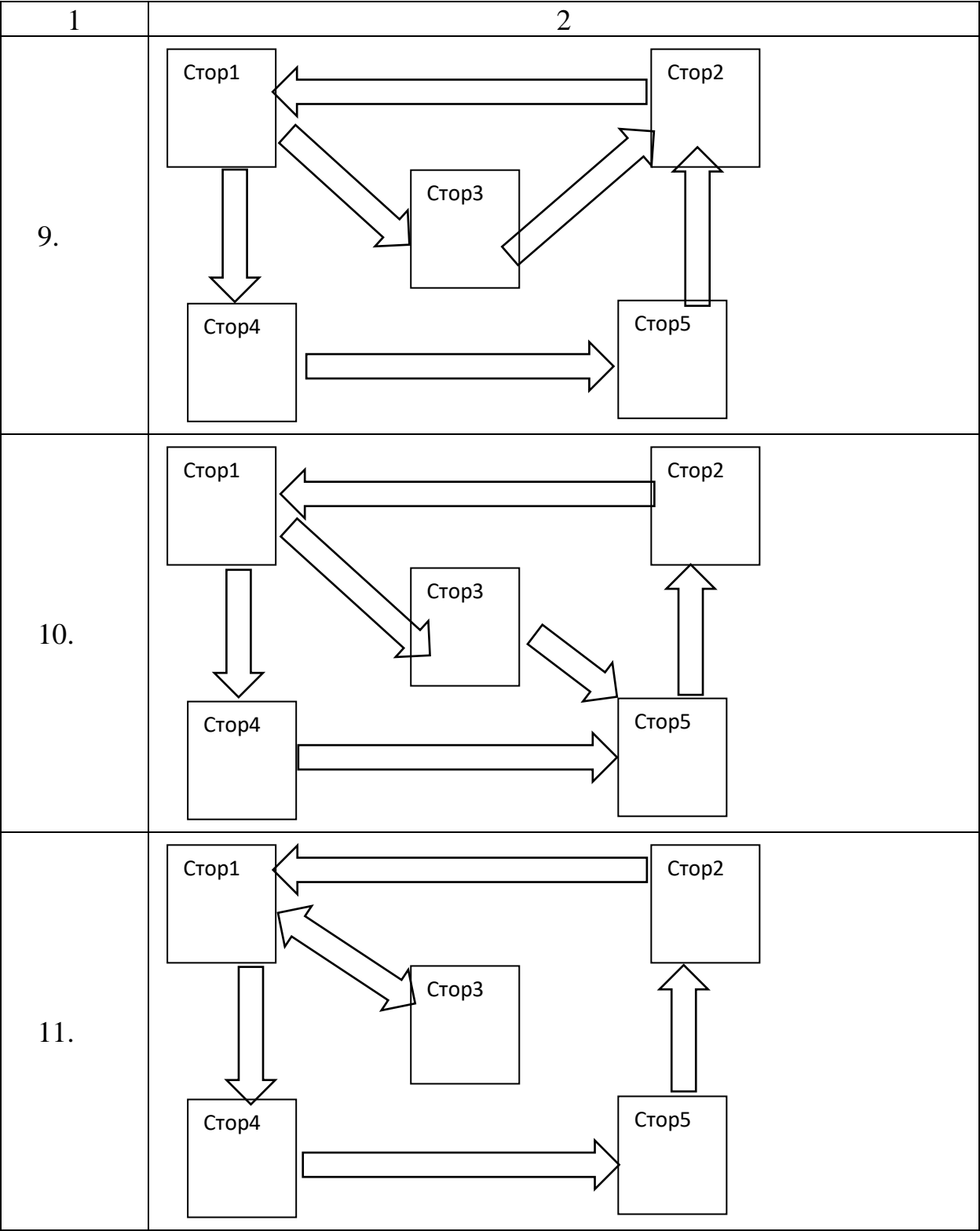
Посилання на зображення задаються з відповідної БД, розміри зображень – з JSP

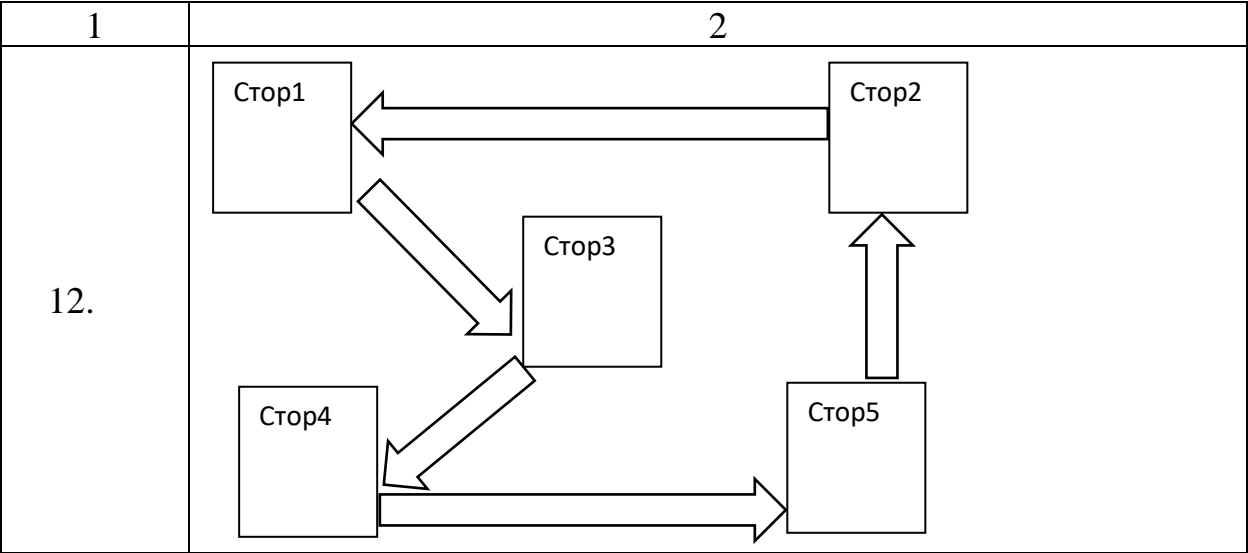
4. Створити JSP, яка викликає сервлет, який в свою чергу «на льоту» створює сторінку із шаховою (без об'єднаних комірок) таблицею із вказаними розмірами. Парні номери варіантів кількість рядків задає на JSP, а кількість стовпчиків – з БД, непарні номери – навпаки. Комірки таблиці послідовно заповнюються цілими числами, починаючи з 1, тобто 1, 2, 3, ...
5. Створити JSP із посиланням на JSP-сайт, який працює за схемою власного варіанту

№ варіанту	Схема переходів
1	2
1.	<pre> graph TD Стор1 <--> Стор3 Стор2 <--> Стор3 Стор4 <--> Стор3 Стор5 <--> Стор3 </pre>
2.	<pre> graph TD Стор1 <--> Стор2 Стор2 <--> Стор3 Стор3 <--> Стор4 Стор4 <--> Стор5 Стор5 <--> Стор1 Стор1 <--> Стор4 Стор2 <--> Стор3 Стор3 <--> Стор4 Стор4 <--> Стор5 Стор5 <--> Стор1 </pre>

1	2
3.	
4.	
5.	

1	2
6.	<pre>graph LR; S1[Стоп1] <--> S2[Стоп2]; S1 <--> S3[Стоп3]; S2 <--> S3; S3 <--> S4[Стоп4]; S3 <--> S5[Стоп5]; S1 --> S4;</pre>
7.	<pre>graph LR; S1[Стоп1] --> S2[Стоп2]; S2 --> S3[Стоп3]; S3 --> S5[Стоп5]; S3 --> S4[Стоп4]; S4 --> S5; S5 --> S1;</pre>
8.	<pre>graph LR; S1[Стоп1] <--> S2[Стоп2]; S1 <--> S3[Стоп3]; S2 <--> S3; S3 <--> S4[Стоп4]; S3 <--> S5[Стоп5]; S1 --> S4;</pre>





Застосування бібліотеки JSTL (Java Standard Tags Library) для реалізації інтерфейсної логіки JSP

Мета: ознайомитися із можливостями бібліотеки JSTL придбати практику у використанні бібліотеки JSTL для оптимального формування та обробки інформації на JSP.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО.

Коротка теоретична довідка:

Стандартна бібліотека тегів JSP (JSTL) представляє набір тегів для спрощення розробки JSP.

Перевага JSTL:

- **швидка розробка**, JSTL надає багато тегів, які спрощують JSP;
- **повторне використання**, коду можна використовувати теги JSTL на різних сторінках;
- **немає необхідності використовувати скриптлети** (`<% ... java-код ... %>`), це дозволяє уникнути скриплетів.

JSTL надає п'ять типів тегів:

1. **Основні теги.** Основний тег JSTL забезпечує підтримку змінних, керування URL-адресами, керування потоком тощо. URL-адреса основного тегу – **<http://java.sun.com/jsp/jstl/core>**. Префікс основного тегу – **c**.
2. **Функціональні теги.** Функціональні теги забезпечують підтримку маніпулювання рядками та довжиною рядків. URL для тегів функцій – **<http://java.sun.com/jsp/jstl/functions>**, а префікс – **fn**.
3. **Теги форматування.** Теги форматування забезпечують підтримку форматування повідомлень, форматування чисел і дати тощо. URL-адреса тегів форматування – **<http://java.sun.com/jsp/jstl/fmt>**, а префікс – **fmt**.
4. **Теги XML.** Теги XML забезпечують керування потоком, перетворення тощо. URL-адреса тегів XML – **<http://java.sun.com/jsp/jstl/xml>**, а префікс – **x**.
5. **Теги SQL.** Теги JSTL SQL забезпечують підтримку SQL. URL-адреса тегів SQL – **<http://java.sun.com/jsp/jstl/sql>**, а префікс – **sql**.

Для підключення JSTL до maven web-проекту можна використати одну з наступних залежностей (dependencies):

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

для javax

```
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.0</version>
</dependency>
```

для jakarta

А на JSP додати тег

```
<% @ taglib uri="шлях" prefix="префікс" %>
```

де шлях та префікс – параметри відповідної групи тегів

Опис найбільш поширених тегів JSTL:

- **c:out** Він відображає результат виразу, подібно до того, як працює тег `<%=...%>`;
- **c:set** Встановлює результат виразу, що оцінюється, у змінній 'scope';
- **c:if** Це умовний тег, який використовується для перевірки умови та відображення основного вмісту, лише якщо вираз обчислюється істинно;
- **c:choose, c:when, c:otherwise** Це простий умовний тег, який включає його основний вміст, якщо оцінена умова є істинною;
- **c:forEach** Це базовий тег ітерації. Він повторює вміст вкладеного тіла фіксовану кількість разів або протягом колекції;
- **sql:setDataSource** Використовується для створення простого джерела даних, придатного лише для створення прототипу;
- **sql:query** Використовується для виконання SQL-запиту, визначеного в його атрибуті sql або в тілі;
- **sql:update** Використовується для виконання оновлення SQL, визначеного в атрибуті sql або в тілі тегу;
- **sql:param** Використовується для встановлення параметра в операторі SQL на вказане значення;
- **sql:dateParam** Використовується для встановлення параметра в операторі SQL на вказане значення `java.util.Date`.
- **sql:transaction** Використовується для забезпечення дії вкладеної бази даних із загальним з'єднанням..

З більш детальною інформацією та з рештою тегів можна ознайомитися, наприклад за посиланням: <https://www.javatpoint.com/jstl-core-tags>

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям. Повертатися із сервлета на JSP треба за допомогою об'єкту класу RequestDispatcher.
2. Виконати друге завдання минулої роботи згідно свого варіанту, але кінцева сторінка має формуватися не «на льоту» сервлетом, а за допомогою відповідних тегів (set, out) бібліотеки JSTL.
3. Виконати третє завдання минулої роботи згідно свого варіанту, але кінцева сторінка має формуватися не «на льоту» сервлетом, а за допомогою відповідних тегів (set, out) бібліотеки JSTL.
4. Виконати четверте завдання минулої роботи згідно свого варіанту, але кінцева сторінка має формуватися не «на льоту» сервлетом, а за допомогою відповідних тегів (set, out) бібліотеки JSTL.
5. Зробити зв'язку JSP-JSP, яка вирішує таку задачу: на першій JSP в окремому текстовому полі ввести рядок, що містить не менше п'яти слів, а в інше окреме поле – кількість разів для повторення обробленого (теги функцій JSTL) за власним варіантом рядка; друга JSP має обробити рядок отриманий з першої JSP та повторити його задану кількість разів

№ варіанту	Функція обробки рядка
1.	Перевести увесь рядок до верхнього регістру
2.	Перевести увесь рядок до нижнього регістру
3.	Виділити перше слово рядка
4.	Виділити останнє слово рядка
5.	Позбутися усіх зайвих пробілів (один пробіл між словами)
6.	Переставити слова рядка у зворотному порядку
7.	Прибрати два останні слова з рядка
8.	Прибрати два перші слова з рядка
9.	Прибрати другу половину рядка
10.	Прибрати першу половину рядка
11.	Перевести друге та передостаннє слово рядка до верхнього регістру
12.	Перевести другу половину рядка до верхнього регістру

6. В робочій базі даних створити таблицю за наступним зразком

e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

Кількість записів можна збільшувати, дані в полях змінювати за потребою.

Зробити зв'язку JSP-JSP, яка вирішує таку задачу: на першій JSP в окремих полях даються параметри запиту за власним варіантом до створеної таблиці; друга JSP має відобразити результат запиту за допомогою SQL-тегів JSTL

№ варіанту	Приблизна форма запиту
1.	Перші N записи
2.	Будь-яке обмеження за віком
3.	Будь-яке обмеження за статтю
4.	Будь-яке обмеження за спеціальністю
5.	Будь-яке обмеження за віком та статтю
6.	Будь-яке обмеження за зарплатою
7.	Будь-яке обмеження за зарплатою та статтю
8.	Будь-яке обмеження за спеціальністю та статтю
9.	Будь-яке обмеження за спеціальністю та віком
10.	Будь-яке обмеження за зарплатою та віком
11.	Усю таблицю із останніми трьома полями
12.	Усю таблицю із першими трьома полями

7. Зробити зв'язку JSP-Servlet-JSP, яка вирішує таку задачу: на першій JSP в окремих полях даються параметри запиту за варіантом минулого завдання до створеної таблиці; Servlet зчитує вміст таблиці, перетворює його на колекцію(ArrayList чи LinkedList) відповідних об'єктів та передає на другу JSP; друга JSP має відобразити результат запиту за допомогою умовних та циклічних тегів JSTL

Лабораторна робота №3

Використання рівнів доступності (scopes) даних для обміну інформацією під час роботи web-застосунку

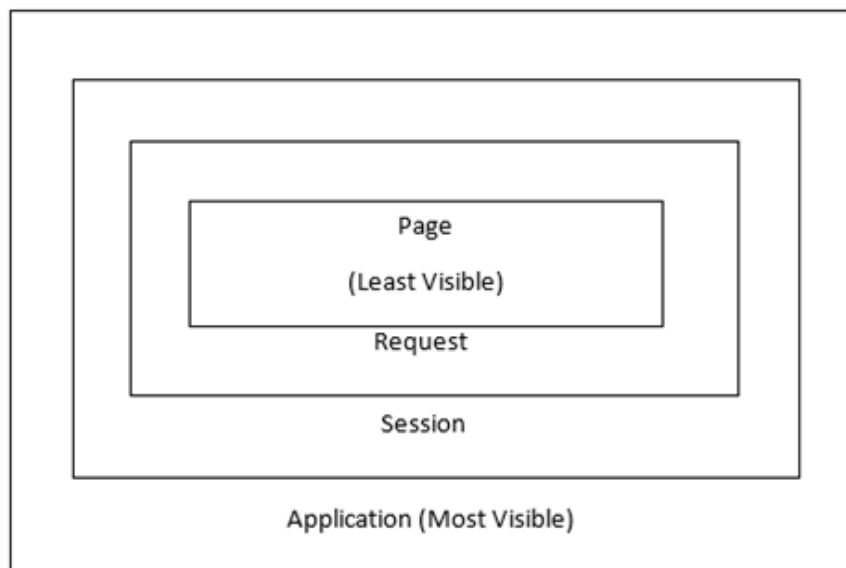
Мета: ознайомитися із різними варіантами визначення змінних щодо рівня їхньої досяжності та способами використання для обміну інформацією у сервлетах та JSP web-застосунку.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО. Завдання, помічені *, виконуються на максимальний оціночний бал

Коротка теоретична довідка:

Схему рівнів визначення змінних можна визначити рисунком



Рівні (scopes) доступу на JSP:

- **Page Scope**

```
<c:set var="name" value="Manisha" scope="page" />
```

...

```
<!--Тільки на даній сторінці -->
```

```
<c:out value="${name}" />
```

- **Request Scope**

```
<c:set var="name" value="Manisha" scope="request" />
```

...

```
<!--На даній сторінці та на усіх, пов'язаних із даною запитом -->
```

```
<c:out value="${name}" />
```

- **Session Scope**

```
<c:set var="name" value="Manisha" scope="session" />
```

...

```
<!--На усіх сторінках чергового сеансу роботи застосунку -->
```

```
<c:out value="${name}" />
```

- **Application Scope**

```
<c:set var="name" value="Manisha" scope="session" />
```

...

```
<!--На усіх сторінках поки не завершена робота застосунку -->
```

```
<c:out value="${name}" />
```

Рівні (scopes) доступу на Servlet:

- **Application Scope**

```
req.getServletContext().setAttribute("name", "application scoped attribute");
```

...

```
String applicationScope = (String)req.getServletContext().getAttribute("a");
```

- **Session Scope**

```
HttpSession session = req.getSession();
```

```
session.setAttribute("name", "session scoped attribute");
```

...

```
HttpSession session = req.getSession();
```

```
String sessionScope = (String)session.getAttribute("b");
```

- **Request Scope**

```
req.setAttribute("name", "request scoped attribute");
```

...

```
String requestScope = (String)req.getAttribute("c");
```

Більш детальну інформацію можна отримати, наприклад, за посиланнями:

<https://dotnettutorials.net/lesson/jsp-scopes/>

https://memorynotfound.com/servlet-attributes-example/#google_vignette

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям. Повертатися із сервлета на JSP треба за допомогою об'єкту класу RequestDispatcher.
2. Створити зв'язку JSP1-Servlet1-JSP2-Servlet2-JSP3-Servlet3-JSP4. Де JSP1, JSP2, JSP3 задають відповідні сторони трикутника, Servlet1, Servlet2, Servlet3 зберігають значення сторін у потрібному «скопі» досяжності та реалізують навігацію між сторінками. На JSP4 розраховується параметр трикутника згідно власного варіанту

№ варіанту	Функція розрахунку параметру трикутника
1.	Визначення периметру
2.	Визначення площі
3.	Визначення середньої довжини сторони
4.	Пошук максимальної сторони
5.	Пошук мінімальної сторони
6.	Визначення коректності вводу сторін (нерівність трикутника)
7.	Визначення, чи є трикутник рівнобедреним
8.	Визначення, чи є трикутник рівнобічним
9.	Визначення, чи є трикутник прямокутним
10.	Визначення кількість сторін, що не перевищують середньої довжини
11.	Визначення кількість сторін, що перевищують середню довжину
12.	Визначити наявність чи відсутність прямого кута у трикутнику

3. Виконати минуле завдання за схемою JSP1–JSP2-JSP3-JSP4, тобто без використання сервлетів.
4. *Виконати друге завдання за допомогою однієї JSP, яка змінює свій інтерфейс залежності від кількості введених сторін та одного сервлета, який зберігає чергову сторону трикутника чи розраховує його параметр теж в залежності від кількості введених сторін
5. Створити чотирьох етапний web-процес реєстрації абстрактного акаунту. На кожному етапі реалізується запит на інформацію згідно до власного варіанту. Після останнього етапу інформація має бути записана до відповідної таблиці бази даних

№ варіанту	1 етап	2 етап	3 етап	3 етап
1.	Email	Login	Password	Address
2.	Login	Email	Address	Password
3.	Address	Login	Email	Address
4.	Address	Password	Login	Email
5.	Password	Address	Email	Login
6.	Password	Email	Address	Login
7.	Email	Login	Address	Password
8.	Password	Email	Login	Address
9.	Password	Login	Email	Address
10.	Address	Login	Password	Email
11.	Login	Address	Email	Password
12.	Password	Email	Address	Login

6. *Виконати минуле завдання за допомогою однієї JSP, яка змінює свій інтерфейс залежності від кількості введених даних майбутнього акаунту та одного сервлета, який зберігає чергову порцію інформації акаунту в залежності від введеної інформації та додає її до відповідної таблиці
7. Створити двох етапний web-процес пошуку абстрактного акаунту. На кожному етапі реалізується запит на інформацію згідно до власного варіанту. Після останнього етапу інформація має бути знайдена у відповідній таблиці бази даних та виведена на окремій JSP

№ варіанту	1 етап	2 етап
1	2	3
1.	Email	Login
2.	Login	Email
3.	Address	Login
4.	Address	Password
5.	Password	Address

1	2	3
6.	Password	Email
7.	Email	Login
8.	Password	Email
9.	Password	Login
10.	Address	Login
11.	Login	Address
12.	Password	Email

8. *Виконати минуле завдання за допомогою однієї JSP, яка змінює свій інтерфейс залежності від кількості введених даних майбутнього акаунту та одного сервлета, який зберігає чергову порцію інформації акаунту в залежності від введеної інформації та шукає її до відповідної таблиці

Лабораторна робота №4

Використання файлів під час роботи web-застосунку

Мета: ознайомитися із різними варіантами використання файлів різних видів під час роботи web-застосунку.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО. Завдання, помічені *, виконуються на максимальний оціночний бал Сторінки кожного завдання групувати у відповідних папках, а сервлети – в пакетах

Коротка теоретична довідка:

Три методи шляху до файлу класу File.

- **getAbsolutePath()** - повертає абсолютний рядок цього абстрактного шляху.
- **getCanonicalPath()** - повертає рядок канонічного шляху цього абстрактного шляху.
- **getPath()** - перетворює цей абстрактний шлях у рядок шляху.

Можна створити екземпляр файлу в Servlet як:

```
File file = new File("test");
```

Буде отримано getAbsolutePath(),getCanonicalPath() і getPath в сервлеті:

- getAbsolutePath() отримає **D:\java_ide\test**
- getCanonicalPath() отримає **D:\ java_ide \test**
- getPath() отримає **test**.

Де **D:\java_ide\test** — папка, де встановлена інтегрована система програмування мовою Java

Якщо проект розгорнуто на автономному tomee:

- getAbsolutePath() отримає **C:\apache-tomee\webapps\project\test**
- getCanonicalPath() отримає **C:\apache-tomee\webapps\project\test**
- getPath() отримає **test**.

Де **C:\apache-tomee\webapps\project\test** — папка, де встановлений локальний сервер **tomee**.

Якщо треба записати файл у каталозі webapp, який буде кореневою папкою проекту (наприклад, **DynaServletProject**) використовуються **ServletContext().getContextPath()** і **ServletContext().getRealPath**, наприклад:

- **getServletContext().getContextPath()** буде отримано
/DynaServletProject
- **getServletContext().getRealPath("/")** буде отримано
D:\workspace\ \webapps\DynaServletProject
- **getServletContext().getRealPath("/WEB-INF")** буде отримано
D:\workspace\webapps\DynaServletProject\WEB-INF

Нижче наведено відповідні значення, коли сервлет запущений із автономного Apache Tomcat:

- **getServletContext().getContextPath()**
/DynaServletProject
- **getServletContext().getRealPath("/")**
D:\apache-tomcat-7.0.33\webapps\DynaServletProject
- **getServletContext().getRealPath("/WEB-INF")**
D:\apache-tomcat-7.0.33\webapps\DynaServletProject\WEB-INF

Можна використовувати **getRealPath("/")**, щоб отримати справжній шлях до кореневої папки.

Запис, наприклад, до XML-файлу всередині webapp кореневої папки з сервлета виконується за допомогою такого коду:

```
String contextPath = getServletContext().getRealPath("/");
```

```
String xmlFilePath=contextPath+"\\test";
```

```
System.out.println(xmlFilePath);
```

```
File myfile = new File(xmlFilePath);
```

```
myfile.createNewFile();
```

Більш докладну інформацію із цього питання можна отримати за наступними посиланнями:

<https://www.baeldung.com/java-path>

<https://zetcode.com/articles/javaxservletservletxml/>

<https://kodejava.org/how-do-i-read-text-file-in-servlet/>

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям. Повертатися із сервлета на JSP треба за допомогою об'єкту класу RequestDispatcher.
2. Провести тестування отримання абсолютного та канонічного шляхів до абстрактного файлу, наприклад, з ім'ям test_file у папках / (коренева) та /WEB-INF/ із застосуванням стандартного класу File. Шляхи мають відображатися на відповідній JSP-сторінці. Під час формування початкового шляху рекомендується застосовувати символ батьківської папки (..) для створення відносного шляху.
3. У папці WEB-INF вручну створити текстовий файл simple.txt таким чином, щоб у ньому було не менш ніж п'ять рядків тексту. За допомогою розроблених web-засобів (JSP, servlet) відобразити на окремій сторінці вміст цього файлу згідно до завдання власного варіанту

№ варіанту	Завдання до відображення вмісту файлу
1	2
1.	Усі рядки файлу у зворотному порядку
2.	Усі парні рядки файлу у зворотному порядку
3.	Усі непарні рядки файлу у зворотному порядку
4.	Усі парні рядки файлу у прямому порядку
5.	Усі непарні рядки файлу у прямому порядку
6.	Перші половини парних рядків файлу у зворотному порядку
7.	Перші половини непарних рядків файлу у зворотному порядку
8.	Перші половини парних рядків файлу у прямому порядку
9.	Перші половини непарних рядків файлу у прямому порядку
10.	Перші половини рядків файлу у зворотному порядку

1	2
11.	Перші половини рядків файлу у прямому порядку
12.	Другі половини рядків файлу у зворотному порядку

4. *У папці WEB-INF вручну створити три текстові файли simple1.txt, simple2.txt, simple3.txt таким чином, щоб у кожному з них було не менш ніж п'ять рядків тексту. За допомогою розроблених web-засобів (JSP, servlet) відобразити на окремій сторінці вміст цих файлів згідно до завдання власного варіанту

№ варіанту	Завдання до відображення вмісту файлів
1	2
1.	«Перемішати» рядки (1, 1, 1, 2, 2, 2, 3, 3. 3. ...) з кожного файлу, починаючи з початку файлу
2.	«Перемішати» рядки (1, 1, 1, 2, 2, 2, 3, 3. 3. ...) з кожного файлу, починаючи з кінця файлу
3.	«Перемішати» перші половини рядків (1/2, 1/2, 1/2, 2/2, 2,2, 2,2, 3/2, 3/2. 3/2. ...) з кожного файлу, починаючи з початку файлу
4.	«Перемішати» перші половини рядків (1/2, 1/2, 1/2, 2/2, 2,2, 2,2, 3/2, 3/2. 3/2. ...) з кожного файлу, починаючи з кінця файлу
5.	simple1/2 + simple2/2 + simple3 + simple2/2+ simple1/2
6.	simple2/2 + simple3/2 + simple1 + simple3/2+ simple2/2
7.	simple1/2 + simple3/2 + simple2 + simple3/2 + simple1/2
8.	simple1/2 + simple2/2 + simple3/2+ Simple1/2 + simple2/2 + simple3/2
9.	simple3/2 + simple2/2 + simple1/2+ imple3/2 + simple2/2 + simple1/2
10.	simple2/2 + simple3/2 + simple1/2+ Simple2/2 + simple3/2 + simple1/2
11.	simple1/2 + simple2/2 + simple3 + simple1/2+ simple2/2
12.	simple2/2 + simple3/2 + simple1 + simple2/2+ simple3/2
13.	simple1/2 + simple3/2 + simple2 + simple1/2 + simple2/2

5. За допомогою web-засобів створити діалог, який пропонує ввести фрагмент тексту, після чого на окремій сторінці відображає його у вигляді блоку із параметрами згідно до свого варіанту, які задаються в окремому прор-файлі, наприклад з ім'ям **configuration.properties** папки WEB-INF

№ варіанту	Параметри блоку у прор-файлі
1	2
1.	Колір тексту, фону, розмір тексту, локація блоку
2.	Колір тексту, фону, розмір тексту, кут нахилу блоку
3.	Колір тексту, фону, розмір тексту, вигляд бордюру блоку
4.	Колір тексту, фону, розмір тексту, радіус бордюру блоку
5.	Колір тексту, фону, розмір тексту, розміри блоку
6.	Колір тексту, фону, розмір тексту, кут повороту блоку

№ варіанту	Місце додання дати	Місце додання часу
1	2	3
1.	Верхній лівий кут	Верхній правий кут
2.	Верхній правий кут	Верхній лівий кут
3.	Нижній лівий кут	Нижній правий кут
4.	Нижній правий кут	Нижній лівий кут
5.	Верхній лівий кут	Нижній лівий кут
6.	Нижній лівий кут	Верхній лівий кут
7.	Верхній правий кут	Нижній правий кут
8.	Нижній правий кут	Верхній правий кут
9.	Верхній лівий кут	Нижній правий кут
10.	Нижній правий кут	Верхній лівий кут
11.	Нижній лівий кут	Верхній правий кут
12.	Верхній правий кут	Нижній лівий кут

8. Перевірити наявність в базі даних таблиці з 6-го завдання другої лабораторної роботи. У випадку відсутності створити її. Розробити комплекс web-засобів, які створюють XML-файл на основі даних цієї таблиці. Вигляд файлу має відповідати власному варіанту. Вміст створеного файлу має бути прочитаний і поданий у вигляді таблиці за допомогою технології AJAX фінальної JSP-сторінки

№ варіанту	Вигляд XML-файлу
1	2
1.	Теговий
2.	Атрибутний
3.	Змішаний
4.	Теговий
5.	Атрибутний
6.	Змішаний
7.	Теговий
8.	Атрибутний
9.	Змішаний
10.	Теговий
11.	Атрибутний
12.	Змішаний

Створення та використання власних налаштованих («кастомних» від англійського custom) java-тегів на JSP

Мета: опанувати принципи створення та застосування кастомних java-тегів різних видів та бібліотек тегів під час роботи web-застосунку.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО. Завдання, помічені *, виконуються на максимальний оціночний бал. Сторінки кожного завдання групувати у відповідних папках, а сервлети – в пакетах. Усі створені теги об'єднати в бібліотеку за допомогою TLD -файлу та розміщення у відповідних пакетах. Для демонстрації роботи кастомних тегів створюються відповідні JSP з відповідним контентом.

Теоретична довідка:

Кастомні теги — це визначені розробником теги дій, які можна використовувати на сторінках сервера Java. Для реалізації операцій з кожним тегом пов'язаний обробник тегів. Таким чином, він відокремлює бізнес-логіку від JSP і допомагає уникнути використання тегів скриплетів. Скриплет вбудовує код Java у саму сторінку JSP, що робить сторінку важкозрозумілою, тому краще уникати використання скриплетів.

Як скриплети, так і кастомні теги використовують сервлети в основі, тому вони не дуже відрізняються за продуктивністю. Але користуальницькі теги завжди мають перевагу перед скриплетами з наступних причин:

- кастомні теги відокремлюють бізнес-логіку від сторінок сервера Java. JSP все ще контролює потік, але обробку делеговано окремому класу Java (класу обробки тегів);
- це підвищує читабельність сторінок сервера Java. JSP без сценаріїв набагато легше читати та розуміти порівняно з JSP, вбудованими зі скриплетами;
- інтерфейсному розробнику простіше використовувати теги, ніж писати код Java всередині JSP;
- кастомні теги можна повторно використовувати в різних програмах, але неможливо повторно використовувати код Java, вбудований у JSP, за допомогою скриплетів.

Синтаксис кастомних тегів:

- порожній кастомний тег (без тіла)

<prefix : suffix attribute = “value”/>

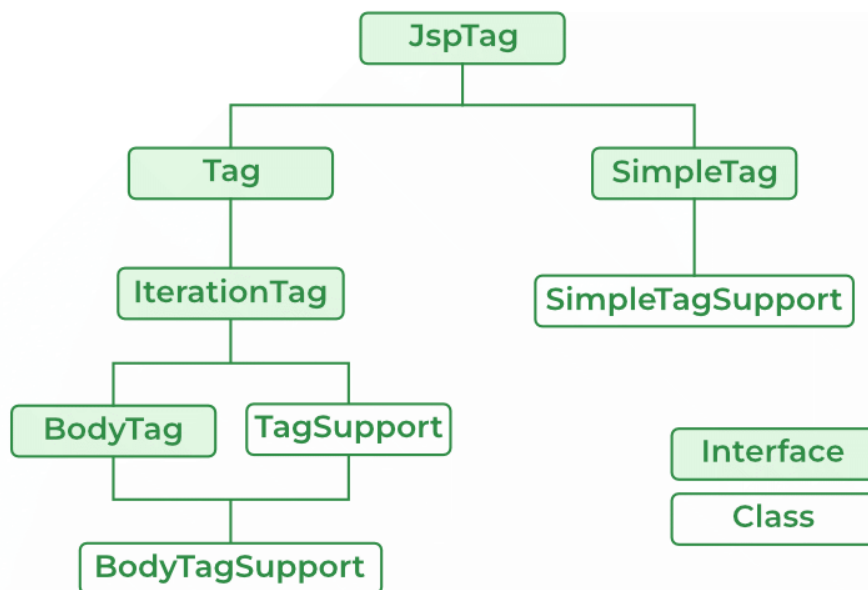
- непорожній кастомний тег

<prefix : suffix attribute = “value”>body</prefix : suffix>

Для розробки спеціальних тегів потрібні наступні три компоненти:

- обробник тегів;
- файл TLD (tags library descriptor дескриптор бібліотеки тегів);
- директива Taglib у файлі jsp.

Обробник тегів -це об'єкт, керований контейнером, створений контейнером JSP під час виконання. Клас Java використовується для реалізації логіки обробки для тегу. Він може містити певні властивості, що відповідають атрибутам або тілу тегу. Усі обробники тегів мають властивість `pageContext` для сторінки JSP, де розміщено тег, і батьківську властивість для обробника тегів до найближчого батьківського тегу. Він повинен реалізовувати/розширювати один із наступних інтерфейсів/класів, наявних у пакеті `javax.servlet.jsp.tagext` чи `jakarta.servlet.jsp.tagext`:



Життєвий цикл обробника тегів:

- під час виконання JSP-контейнер відповідає за створення об'єкта реалізації класу обробника тегів за допомогою конструктора `no-arg`;
- після створення об'єкта контейнер JSP викликає методи `setParent` і `setPageContext` в обробнику тегів;

- контейнер JSP потім викликає методи встановлення в обробнику тегів, щоб установити його властивості, використовуючи значення відповідних атрибутів тегу;
- коли об'єкт готовий до використання, для нього викликаються різні методи життєвого циклу;
- об'єднанням обробників тегів керує контейнер JSP. Той самий обробник тегів може використовуватися для кількох входжень тегів на сторінці, якщо значення атрибутів однакові. Але якщо значення атрибутів відрізняються, можливо, потрібно скинути властивості. У випадку класів SimpleTag і SimpleTagSupport обробники тегів не об'єднуються.

Інтерфейс **Tag** може бути реалізований обробником тегів, щоб забезпечити логіку обробки, якщо тіло тегу не потрібно використовувати. Він надає такі методи:

- *int doEndTag()*
- *int doStartTag()*
- *Tag getParent()*
- *void release()*
- *void setPageContext(PageContext pc)*
- *void setParent(Tag t)*

doStartTag() обчислюється на початку дії тегу і повертає одне з наступного:

- `static int EVAL_BODY_INCLUDE`
- `static int SKIP_BODY`

doEndTag() обчислюється в кінці дії тегу та повертає одне з наступного:

- `static int EVAL_PAGE`
- `static int SKIP_PAGE`

Якщо сторінку включено іншим сервлетом/JSP, лише поточна сторінка пропускається у випадку, якщо *doEndTag()* повертає `SKIP_PAGE`.

Інтерфейс **IterationTag** може бути реалізований, якщо тіло тегу потрібно повторити певну кількість разів. Він забезпечує наступний метод:

int doAfterBody()

doAfterBody() викликається після кожної ітерації методу тіла, щоб вирішити, чи потрібно повторювати тіло чи ні. Він повертає одне з наступного:

- `static int SKIP_BODY`
- `static int EVAL_BODY_AGAIN`

Якщо повертається `SKIP_BODY`, контейнер викликає метод *doEndTag()* і продовжує роботу далі. Якщо повертається `EVAL_BODY_AGAIN`, тіло обробляється повторно.

Клас **TagSupport** - це базовий клас, який реалізує інтерфейси **Tag** і **IterationTag**. Його можна використовувати для реалізації обробника тегів, якщо реалізація для всіх методів не потрібна. Він забезпечує стандартну реалізацію всіх абстрактних методів своїх батьківських інтерфейсів.

Інтерфейс **BodyTag** може бути реалізований обробником тегів, якщо потрібно оцінити тіло тегу. Він надає такі методи:

- `void doInitBody()`
- `void setBodyContent(BodyContent b)`

На відміну від `IterationTag`, `doStartTag()` методу `BodyTag` може повертати одне з наступного:

- `static int EVAL_BODY_INCLUDE`
- `static int SKIP_BODY`
- `static int EVAL_BODY_BUFFERED`

Якщо метод `doStartTag()` повертає `EVAL_BODY_BUFFERED`, створюється об'єкт `BodyContent` і передається в метод `setBodyContent()`. `BodyContent`: це підклас `JspWriter`. Нижче наведено деякі корисні методи класу `BodyContent`.

- `void clearBody()`
- `Reader getReader()`
- `String getString()`
- `void writeOut(Writer out)`

Ці методи `BodyContent` можна використовувати для перетворення його вмісту в `String`, для читання його вмісту та очищення вмісту.

Клас **BodyTagSupport** - це базовий клас, який реалізує клас `BodyTag` і розширює клас `TagSupport`. Він забезпечує стандартну реалізацію своїх батьківських інтерфейсів.

Обробник тегів може реалізувати інтерфейс **SimpleTag**, щоб забезпечити логіку обробки. Він має набагато простіший життєвий цикл порівняно з класичними обробниками тегів. Інтерфейс `SimpleTag` надає такі методи:

- `void doTag()`
- `JspTag getParent()`
- `void setJspBody(JspFragment jspBody)`
- `void setJspContext(JspContext pc)`
- `void setParent(JspTag parent)`

Метод `doTag()` використовується для забезпечення всієї логіки обробки. Він виконується лише один раз, тому логіка ітерації, якщо потрібно, повинна бути інкапсульована в ньому.

Метод `setJspBody()` встановлює тіло тегу, якщо його потрібно використати. Він приймає об'єкт `JspFragment`.

`JspFragment`: інкапсулює частину коду JSP в об'єкт, який можна викликати так часто, як це необхідно. Визначення фрагмента JSP має містити лише текст шаблону та елементи дії JSP. Іншими словами, він не повинен містити скриплетів або вирази скриплетів. Клас `JspFragment` надає такі методи:

- `JspContext getJspContext()`
- `void invoke(Writer out)`

Клас **SimpleTagSupport** - це базовий клас інтерфейсу `SimpleTag`. Його можна розширити, щоб надати код обробника. З наведеного вище обговорення

можна зробити висновок, що обробник тегів повинен реалізувати один із таких інтерфейсів: Tag, IterationTag, SimpleTag або розширити будь-який із наступних класів: TagSupport, BodyTagSupport, SimpleTagSupport, щоб забезпечити логіку обробки для тегу.

TLD (дескриптор бібліотеки тегів) - це файл, збережений із розширенням .tld, який містить набір пов'язаних тегів, зіставлених із відповідними обробниками тегів разом із їхнім описом, таким як ім'я тегу, атрибути тегу тощо. Кореневий тег документа TLD – <taglib>, у якому кілька тегів можна інкапсулювати за допомогою тегу <tag>. Він зберігається в папці WEB-INF. Він складається з наступного заголовка:

```
<taglib version="2.0" xmlns="http://java.sun.com/xml/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-
jsptaglibrary_2_0.xsd">
```

<Taglib> складається з таких піделементів:

- <tlib-version>: визначає версію бібліотеки тегів;
- <jsp-version>: визначає версію JSP;
- <uri>: надає URI для бібліотеки тегів.

<Tag> складається з таких піделементів:

- <body-content>: описує тип основного вмісту: порожній, JSP, без сценаріїв, залежний від тегів;
- <name>: використовується для надання унікальної назви тегу. (суфікс);
- <tagclass>: використовується для надання класу обробника тегу для тегу;
- <attribute>: використовується для надання атрибутів для тегу.

<Attribute> додатково інкапсулює такі властивості:

- <name>: використовується для надання унікального імені атрибуту.
- <required>: використовується для визначення того, обов'язковий чи необов'язковий атрибут. Приймає логічне значення.
- <rtexprvalue>: використовується для вказівки, чи можна додавати вираз часу виконання до значення атрибуту. Приймає логічне значення.
- <type>: використовується для визначення типу значення, яке буде надано атрибуту. За замовчуванням прийнято значення String.
- <fragment>: використовується для визначення того, чи атрибут зберігає JspFragment чи ні.

TLD разом із обробниками тегів утворюють бібліотеку тегів, яка потім розповсюджується та використовується в JSP. Стандартним прикладом таких бібліотек є JSTL (стандартна бібліотека тегів Java), яка надає кілька корисних тегів для потоку керування, обробки рядків, операцій з базою даних тощо.

Директива **Taglib** використовується для доступу до певної бібліотеки тегів у JSP. Він визначає URI бібліотеки тегів і її префікс.

Синтаксис: `<%@taglib uri=" " prefix=" " %>`

Додаткову інформацію можна отримати за наступними посиланнями:

<https://www.geeksforgeeks.org/custom-tags-in-jsp/>

https://www.tutorialspoint.com/jsp/jsp_custom_tags.htm

<https://www.javatpoint.com/custom-tags>

<https://www.geeksforgeeks.org/custom-tags-with-body-in-jsp/>

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям. Повертатися із сервлета на JSP треба за допомогою об'єкту класу RequestDispatcher.
2. Створити простий тег, який формує повідомлення, що містить власне прізвище, ім'я та по-батькові, зовнішній вигляд яких відповідає варіанту

№ варіанту	Інформація	Розмір (см)	Колір	Тип вирівнювання	Форма
1	2	3	4	5	6
1.	Прізвище	7	Жовтий	Центр	Жирний
	Ім'я	5	Зелений	Ліве	Курсив
	По-батькові	1	Червоний	Праве	3 підкресленням
2.	Прізвище	2	Сірий	Ліве	Жирний, курсив
	Ім'я	7	Коричневий	Центр	3 підкресленням
	По-батькові	3	Рожевий	Праве	Жирний
3.	Прізвище	2	Пурпурний	Ліве	Курсив, з підкресленням
	Ім'я	5	Блакитний	Праве	Жирний, курсив, з підкресленням
	По-батькові	7	Синій	Центр	3 підкресленням
4.	Прізвище	1	Фіолетовий	Центр	3 підкресленням
	Ім'я	6	Жовтий	Центр	Жирний, курсив, з підкресленням
	По-батькові	5	Червоний	Ліве	Жирний

1	2	3	4	5	6
5.	Прізвище	7	Зелений	Праве	Жирний, з підкресленням
	Ім'я	6	Синій	Ліве	Курсив, з підкресленням
	По-батькові	5	Блакитний	Ліве	Жирний, курсив
6.	Прізвище	1	Коричневий	Центр	Курсив,
	Ім'я	2	Рожевий	Праве	3 підкресленням
	По-батькові	3	Фіолетовий	Центр	Жирний,
7.	Прізвище	2	Сірий	Ліве	Жирний, з підкресленням
	Ім'я	3	Коричневий	Праве	Курсив, з підкресленням
	По-батькові	7	Жовтий	Ліве	Жирний, курсив, з підкресленням
8.	Прізвище	2	Зелений	Центр	Жирний, курсив, з підкресленням
	Ім'я	3	Сірий	Праве	3 підкресленням
	По-батькові	4	Рожевий	Праве	Жирний,
9.	Прізвище	4	Фіолетовий	Ліве	Курсив
	Ім'я	3	Зелений	Ліве	Жирний
	По-батькові	1	Рожевий	Центр	3 підкресленням
10.	Прізвище	5	Жовтий	Праве	Курсив, з підкресленням
	Ім'я	7	Червоний	Центр	Жирний, з підкресленням
	По-батькові	3	Зелений	Праве	Жирний, курсив, з підкресленням
11.	Прізвище	7	Синій	Центр	3 підкресленням
	Ім'я	1	Зелений	Ліве	Курсив, з підкресленням
	По-батькові	2	Красний	Центр	Жирний, курсив, з підкресленням
12.	Прізвище	6	Рожевий	Ліве	Жирний, курсив
	Ім'я	4	Синій	Праве	Курсив, з підкресленням
	По-батькові	7	Блакитний	Центр	Жирний, курсив, з підкресленням

3. Створити простий тег, який формує поточну дату у форматі згідно до власного варіанту

№ варіанту	Формат дати	Вигляд місяця
1	2	3
1.	Європейський	Прописом
2.	Американський	Числом
3.	Європейський	Числом
4.	Американський	Прописом
5.	Європейський	Прописом
6.	Американський	Числом
7.	Європейський	Числом
8.	Американський	Прописом
9.	Європейський	Прописом
10.	Американський	Числом
11.	Європейський	Числом
12.	Американський	Прописом

4. Створити простий тег, який формує поточний час у форматі згідно до власного варіанту

№ варіанту	Формат часу	Наявність секунд/мілісекунд
1	2	3
1.	12-годинний	секунди
2.	24-годинний	мілісекунди
3.	12-годинний	секунди, мілісекунди
4.	24-годинний	секунди, мілісекунди
5.	12-годинний	мілісекунди
6.	24-годинний	секунди
7.	12-годинний	секунди
8.	24-годинний	мілісекунди
9.	12-годинний	секунди, мілісекунди
10.	24-годинний	секунди, мілісекунди
11.	12-годинний	мілісекунди
12.	24-годинний	секунди

5. Створити атрибутний тег, який формує зображення за назвою відповідного файлу (файли зображень мають бути в окремій папці), висотою та шириною. Використовуючи створений тег створити топологію зображень згідно до власного варіанту

№ варіанту	Форма топології				
1	2				
1.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
2.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
3.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
4.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
5.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
6.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
7.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
8.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5
9.	Зобр1	Зобр2	Зобр3	Зобр4	Зобр5

1	2
10.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
11.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>
12.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div> <div>Зобр4</div> <div>Зобр5</div>

№ варіанту	Форма топології
1	2
1.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>
2.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>
3.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>
4.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>
5.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>
6.	<div>Зобр1</div> <div>Зобр2</div> <div>Зобр3</div>

1	2		
7.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>
8.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>
9.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>
10.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>
11.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>
12.	<div>Зобр1</div>	<div>Зобр2</div>	<div>Зобр3</div>

6. *Перевірити наявність в базі даних таблиці з 6-го завдання другої лабораторної роботи. У випадку відсутності створити її. Створити атрибутний тег, який зазначеннями атрибутів згідно до власного варіанту робить відповідну вибірку з даної таблиці

№ варіанту	Завдання для визначення атрибутів
1	2
1.	Перші N записи
2.	Будь-яке обмеження за віком
3.	Будь-яке обмеження за статтю
4.	Будь-яке обмеження за спеціальністю
5.	Будь-яке обмеження за віком та статтю
6.	Будь-яке обмеження за зарплатою
7.	Будь-яке обмеження за зарплатою та статтю
8.	Будь-яке обмеження за спеціальністю та статтю
9.	Будь-яке обмеження за спеціальністю та віком
10.	Будь-яке обмеження за зарплатою та віком

1	2
11.	Усю таблицю із останніми трьома полями
12.	Усю таблицю із першими трьома полями

7. Створити body-тег, який повторює своє тіло три рази Перший раз із параметрами «прізвища», другий – «імені», третій – «по-батькові» з таблиці другого завдання даної роботи.

8. Створити body-тег, який адаптує своє тіло згідно власного варіанту

№ варіанту	Тип адаптації
1.	Видаляє зайві пробіли по краях тіла та між словами залишає не більше одного
2.	Усікає тіла до довжини 7 слів зліва, решту відкидає, а замість неї додає «...»
3.	Переводить усі слова першої половини тіла до верхнього регістру
4.	Переводить усі слова другої половини тіла до верхнього регістру
5.	Переводить усі перші символи слів першої половини тіла до верхнього регістру
6.	Переводить усі перші символи слів другої половини тіла до верхнього регістру
7.	Переводить усі перші символи слів тіла до верхнього регістру
8.	Переводить усі парні слова тіла до верхнього регістру
9.	Переводить усі непарні слова тіла до верхнього регістру
10.	Переводить усі останні символи слів тіла до верхнього регістру
11.	Усікає тіла до довжини 7 слів з правого боку, решту відкидає, а замість неї додає «...»
12.	Бере останні три слова з тіла

9. Створити атрибутний body-тег, який у якості атрибута приймає рядок та обробляє своє тіло згідно власного варіанту

№ варіанту	Тип обробки
1.	Лишити у тілі слова, в яких зустрічається атрибутний рядок
2.	Лишити у тілі слова, в яких не зустрічається атрибутний рядок
3.	До кожного слова тіла спереду додати атрибутний рядок
4.	До кожного слова тіла ззаду додати атрибутний рядок
5.	Замінити усі слова тіла атрибутним рядком
6.	Замінити перші три слова тіла атрибутним рядком
7.	Замінити останні три слова тіла атрибутним рядком
8.	Вставити атрибутний рядок в середину тіла
9.	Вставити атрибутний рядок на початку та в кінці тіла
10.	Підрахувати кількість слів, в яких зустрічається атрибутний рядок
11.	Підрахувати кількість слів, в яких не зустрічається атрибутний рядок
12.	Залишити частину тіла до слова, в якому зустрічається атрибутний рядок

10.*Створити атрибутний body-тег, який у якості атрибута приймає ім'я колекції, розміщеній в сесійній області, і друкує у зворотному порядку. Підказка: для звертання до сесійних атрибутів використовується наступний код

```
HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();  
HttpSession session = request.getSession();
```

Лабораторна робота №6

Синтаксичний аналіз (парсинг від англ. parsing – процес синтаксичного аналізу) XML (eXtensible Markup Language) та JSON (Java Script Object Notation) – файлів у web-застосунках

Мета: опанувати можливості сучасних парсерів щодо створення та обробки XML та JSON файлів різної структури.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО. Завдання, помічені *, виконуються на максимальний оціночний бал. Сторінки кожного завдання групувати у відповідних папках, а сервлети – в пакетах. Для створення та парсингу файлів в проекті рекомендується створити відповідні папки, наприклад XML-INF та JSON-INF, у папці webapp та використовувати відповідні шляхи для майбутніх файлів.

Теоретична довідка:

XML — це проста текстова мова, розроблена для зберігання та передачі даних у форматі звичайного тексту. Розшифровується як Extensible Markup Language. Нижче наведено деякі основні особливості XML.

XML — це мова розмітки.

XML — це мова на основі тегів, як HTML.

Теги XML не є попередньо визначеними, як HTML.

Можна визначити власні теги, тому це називається розширюваною мовою.

XML-теги створені для самоопису.

XML — це рекомендація W3C щодо зберігання та передачі даних.

Наприклад:

```
<?xml version = "1.0"?>
```

```
<Class>
```

```
  <Name>First</Name>
```

```
  <Sections>
```

```
    <Section>
```

```
      <Name>A</Name>
```

```
      <Students>
```

```
        <Student>Rohan</Student>
```

```
        <Student>Mohan</Student>
```

```
        <Student>Sohan</Student>
```

```
<Student>Lalit</Student>
<Student>Vinay</Student>
</Students>
</Section>

<Section>
  <Name>B</Name>
  <Students>
    <Student>Robert</Student>
    <Student>Julie</Student>
    <Student>Kalie</Student>
    <Student>Michael</Student>
  </Students>
</Section>
</Sections>
</Class>
```

Переваги

1. Будучи простим текстом, XML не залежить від технологій. Його можна використовувати будь-якою технологією для зберігання та передачі даних.
2. XML використовує простий текстовий формат. Він читабельний і зрозумілий людині.
3. XML користувацькі теги можна створювати та використовувати дуже легко.
4. Можна легко перевірити використання структур XML.

Недоліки

1. Зазвичай XML-файли містять багато термінів, що повторюються.
2. Будучи багатослівною мовою, розмір файлу XML збільшує витрати на передачу та зберігання.

Парсер XML надає спосіб отримати доступ або змінити дані в документі XML. Java надає кілька варіантів парсингу XML-документів.

1. DOM Parser – аналізує XML-документ шляхом завантаження повного вмісту документа та створення його повного ієрархічного дерева в пам'яті.
2. SAX Parser – аналізує XML-документ на основі тригерів на основі подій. Не завантажує весь документ у пам'ять.
3. Синтаксичний аналізатор JDOM – аналізує XML-документ подібно до аналізатора DOM, але простіше.

4. StAX Parser – аналізує XML-документ подібно до аналізатора SAX, але більш ефективним способом.
5. Парсер XPath – аналізує XML-документ на основі виразу та широко використовується разом із XSLT.
6. DOM4J Parser – бібліотека Java для аналізу XML, XPath і XSLT за допомогою Java Collections Framework. Він забезпечує підтримку DOM, SAX і JAXP.
7. Для об'єктно-орієнтованого аналізу XML доступні API JAXB і XSLT.

Не деяких популярних сучасних парсерах слід зупинитися більш детально.

DOM (Document Object Model - об'єктна модель документа) є офіційною рекомендацією Консорціуму Всесвітньої павутини (W3C). Він визначає інтерфейс, який дозволяє програмам отримувати доступ і оновлювати стиль, структуру та вміст документів XML. XML-парсери, які підтримують DOM, реалізують цей інтерфейс.

Коли використовувати:

- коли потрібно багато знати про структуру документа;
- коли потрібно переміщати частини XML-документа (наприклад, можна відсортувати певні елементи);
- інформацію в XML-документі потрібно використовувати кілька разів.

Після аналізу XML-документ за допомогою парсера DOM, отримується деревовидна структура, яка містить усі елементи документа. DOM надає різноманітні функції, які можна використовувати для вивчення вмісту та структури документа.

Перевагою DOM є те, що це загальний інтерфейс для роботи зі структурами документів. Однією з цілей його розробки є те, що код Java, написаний для одного сумісного з DOM парсера, повинен працювати на будь-якому іншому сумісному з DOM парсері без необхідності вносити будь-які зміни.

DOM визначає декілька інтерфейсів Java. Ось найпоширеніші з них:

- Node – базовий тип даних DOM;
- Element – переважна більшість об'єктів, з якими ви матимете справу, є елементами;
- Attr – представляє атрибут елемента;
- Text – фактичний вміст елемента або атрибута;
- Document – представляє весь документ XML. Об'єкт Document часто називають деревом DOM.

Поширені методи DOM:

Document.getDocumentElement() – Повертає кореневий елемент документа.

Node.getFirstChild() – повертає першу дочірню частину даного вузла.

Node.getLastChild() – повертає останній дочірній елемент даного вузла.

`Node.getNextSibling()` – ці методи повертають наступного брата або сестру даного вузла.

`Node.getPreviousSibling()` – ці методи повертають попередній рідний брат даного вузла.

`Node.getAttribute(attrName)` – для певного вузла він повертає атрибут із запитаною назвою.

SAX (Simple API for XML) - це аналізатор XML-документів на основі подій. На відміну від аналізатора DOM, аналізатор SAX не створює дерева аналізу. SAX — це потоковий інтерфейс для XML, що означає, що програми, які використовують SAX, отримують сповіщення про подію XML-документа, який обробляється елементом і атрибутом, у певний момент часу в послідовному порядку, починаючи з верхньої частини документа та закінчуючи закриттям Елемент ROOT.

Читає XML-документ зверху вниз, розпізнаючи маркери, з яких складається добре сформований XML-документ. Токени обробляються в тому ж порядку, що й у документі. Повідомляє прикладній програмі природу токенів, з якими зіткнувся аналізатор, коли вони виникають. Прикладна програма надає обробник "події", який необхідно зареєструвати в аналізаторі. Коли токени ідентифікуються, викликаються методи зворотного виклику в обробнику з відповідною інформацією.

Коли використовувати:

- можна обробляти XML-документ у лінійному порядку зверху вниз;
- документ не є глибоко вкладеним;
- документ дуже великий XML-документ, дерево DOM якого споживатиме забагато пам'яті. Типові реалізації DOM використовують десять байт пам'яті для представлення одного байта XML;
- проблема, яку потрібно вирішити, стосується лише частини XML-документа;
- дані стають доступними, як тільки їх бачить аналізатор, тому SAX добре працює для XML-документа, який надходить через потік.

Недоліки:

- не довільного доступу до XML-документа, оскільки він обробляється лише в прямому порядку;
- якщо потрібно відстежувати дані, які побачив аналізатор, або змінити порядок елементів, ви повинні написати код і зберегти дані самостійно.

Інтерфейс `ContentHandler` визначає методи зворотного виклику, які аналізатор SAX використовує для сповіщення прикладної програми про компоненти документа XML, які вона побачила.

`void startDocument()` – викликається на початку документа.

`void endDocument()` – викликається в кінці документа.

`void startElement(String uri, String localName, String qName, Attributes atts)` – викликається на початку елемента.

`void endElement(String uri, String localName, String qName)` – викликається в кінці елемента.

`void characters(char[] ch, int start, int length)` – викликається, коли зустрічаються символічні дані.

`void ignorableWhitespace(char[] ch, int start, int length)` – Викликається, коли присутній DTD і зустрічається ігнорований пробіл.

`void processingInstruction(String target, String data)` – Викликається, коли розпізнається інструкція обробки.

`void setDocumentLocator(локатор локатора)` – надає локатор, який можна використовувати для визначення позицій у документі.

`void skippedEntity(String name)` – Викликається, коли зустрічається невіршена сутність.

`void startPrefixMapping(String prefix, String uri)` – Викликається, коли визначено нове відображення простору імен.

`void endPrefixMapping(String prefix)` – викликається, коли визначення простору імен закінчує свою область.

Інтерфейс атрибутів

Цей інтерфейс визначає методи обробки атрибутів, пов'язаних з елементом.

`int getLength()` – Повертає кількість атрибутів.

`String getQName(int index)`

`String getValue(int index)`

`String getValue (рядок qname)`

XPath є офіційною рекомендацією Консорціуму Всесвітньої павутини (W3C). Він визначає мову для пошуку інформації у файлі XML. Він використовується для перегляду елементів і атрибутів документа XML. XPath надає різні типи виразів, які можна використовувати для запиту відповідної інформації з документа XML.

Визначення структури – XPath визначає частини XML-документа, такі як елемент, атрибут, текст, простір імен, інструкція обробки, коментар і вузли документа.

Вирази шляху – XPath надає потужні вирази шляху, такі як вибір вузлів або список вузлів у документах XML.

Стандартні функції – XPath надає багату бібліотеку стандартних функцій для маніпулювання рядковими значеннями, числовими значеннями, порівнянням дати й часу, маніпулювання вузлом і QName, маніпулювання послідовністю, логічними значеннями тощо.

Основна частина XSLT – XPath є одним із основних елементів стандарту XSLT, тому для роботи з документами XSLT необхідно мати достатні знання XPath.

Рекомендація W3C – XPath є офіційною рекомендацією World Wide Web Consortium (W3C).

Додаткові парсери **JDOM** а **StAX** розширюють можливості парсерів DOM та SAX відповідно, але для їх використання в проект треба додавати відповідні бібліотеки `jdom.jar` . та `stax.jar` або безпосередньо, або через залежності відповідних програм збирання проектів.

Більш детальну інформацію із відповідними прикладами можна знайти за посиланням

https://www.tutorialspoint.com/java_xml/index.htm

JSON (JavaScript Object Notation) — це легкий, зручний для читання та незалежний від мови формат обміну даними. Спочатку JSON був похідним від JavaScript об'єктів, і де-факто став стандартним форматом для обміну даними між серверами та клієнтами.

JSON має лише дві структури даних: об'єкти та масиви. Об'єкт — це неупорядкований набір із нуля або більше пар ключ-значення. Масив — це список із нуля чи більше значень. Значеннями можуть бути рядки, числа, логічні значення, значення `null`, вкладені об'єкти та масиви.

Наприклад:

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com",
  "age": 32,
  "address": {
    "street": "155 Middleville Road",
    "city": "New York",
    "state": "New York",
    "zipCode": 10045
  },
  "paymentMethods": [
    "PayPal",
    "Stripe"
  ],
  "projects": [
    {
      "title": "Business Website",
      "budget": 45000
    },
    {
      "title": "Sales Dashboard",
```

```
    "budget": 85000
  }
]
}
```

Існують декілька бібліотек парсерів для роботи із JSON-файлами

JSON.simple — це проста бібліотека для обробки даних JSON у Java. Він дозволяє читати, записувати, аналізувати та запитувати JSON у повній відповідності до специфікацій JSON (RFC4627).

Щоб додати JSON.simple до вашого проекту Gradle, треба додати таку залежність до файлу **build.gradle**:

```
implementation 'com.github.cliftonlabs:json-simple:3.1.0'
```

Для Maven додайте наведену нижче залежність у свій файл **pom.xml**:

```
<dependency>
  <groupId>com.github.cliftonlabs</groupId>
  <artifactId>json-simple</artifactId>
  <version>3.1.0</version>
</dependency>
```

Jackson — ще одна популярна бібліотека Java для читання та запису даних JSON. Джексон надає клас ObjectMapper для перетворення об'єктів Java у їх представлення JSON. Його можна використовувати для перетворення об'єкта JSON назад на еквівалентний об'єкт Java.

Просто додайте наступну залежність до файлу build.gradle вашого проекту Gradle, щоб включити підтримку Jackson:

```
implementation 'com.fasterxml.jackson.core:jackson-databind:2.10.0'
```

Для Maven додайте наведену нижче залежність до вашого файлу pom.xml:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.10.0</version>
</dependency>
```

Gson - це ще одна бібліотека Java, розроблена та підтримувана Google, для перетворення об'єктів Java у їх представлення JSON. Ви також можете

використовувати його для перетворення рядка JSON назад на еквівалентний об'єкт Java.

Gson надає прості методи toJson() і fromJson(), за допомогою яких можна легко конвертувати об'єкт Java у JSON і з JSON.

Щоб додати Gson до вашого проекту Gradle, додайте наведену нижче залежність до файлу build.gradle:

```
implementation 'com.google.code.gson:gson:2.8.6'
```

Для Maven додайте таку залежність до свого файлу pom.xml:

```
<dependency>  
<groupId>com.google.code.gson</groupId>  
<artifactId>gson</artifactId>  
<version>2.8.6</version>  
</dependency>
```

Moshi - ще одна потужна бібліотека JSON, створена Square для Kotlin і Java. Це полегшує аналіз JSON на об'єкти Java і конвертування їх назад у представлення JSON. Moshi має вбудовану підтримку для читання та запису основних типів даних Java, включаючи примітиви, колекції, рядки та перерахування.

Якщо ви хочете використовувати Moshi в проекті Gradle, додайте таку залежність до свого файлу build.gradle:

```
implementation 'com.squareup.moshi:moshi:1.8.0'
```

Для Maven додайте наведену нижче залежність до вашого файлу pom.xml:

```
<dependency>  
<groupId>com.squareup.moshi</groupId>  
<artifactId>moshi</artifactId>  
<version>1.8.0</version>  
</dependency>
```

Більш детальну інформацію із відповідними прикладами можна знайти за посиланням

<https://attacomsian.com/blog/java-read-write-json-files>

Для виконання подальших завдань ознайомитися та проаналізувати предметну область згідно власного варіанту

№ варіанту	Предметна область	Сутності та їх поля (властивості). Головні сутності (довідники) виділені, зв'язувальна - підкреслена
1	2	3
1.	Бібліотека	Книга (автор, назва, рік видання, ціна), Читач (ПІБ, адреса, телефон), <u>Реєстраційний запис</u> (читач, книга, дата видачі, дата повернення, дата фактичного повернення)
2.	Товарний склад	Товар (код, назва, кількість, ціна), Постачальник (назва, адреса, телефон, електронна пошта), <u>Замовлення</u> (товар, постачальник, кількість поставки, термін поставки)
3.	Виробництво	Вироби (код виробу, назва виробу, тип виробу: індивідуальний, серійний, крупносерійний, масовий), Виробник (назва, адреса, електронна пошта), <u>Виробництво</u> (вироби, постачальник, річний обсяг випуску, собівартість)
4.	Мережа магазинів	Магазин (номер, назва, торгівельна площа, складська площа), Підприємець (ПІБ, адреса, банк, номер рахунку), Власник магазину (магазин, підприємець, вартість магазину, річний дохід)
5.	Авторемонтна майстерня	Автомобіль (марка, номерний знак, дата надходження), Автомеханік (ПІБ, табельний номер, кваліфікація), <u>Наряд</u> (автомобіль, автомеханік, вид робіт, вартість, термін виконання)
6.	Деканат	Студент (ПІБ, факультет, курс, група), Дисципліна (назва, кількість годин, викладач), <u>Сесія</u> (студент, дисципліна, форма контролю: залік/іспит, оцінка, дата складання)
7.	Організація послуг	Послуга (шифр послуги, найменування, вартість), Замовник (назва, адреса, розрахунковий рахунок), <u>Договір</u> (послуга, замовник, кількість послуг, термін виконання, дата закінчення договору)

1	2	3
8.	Поліклініка	Пацієнт (реєстраційний код, ПІБ, адреса, вік), Лікар (ПІБ, спеціалізація, кваліфікація), <u>Реєстраційний запис</u> (пацієнт, лікар, діагноз, дата звертання до лікаря, результат лікування)
9.	Телефонна мережа	Телефон (номер, дата встановлення, наявність блокатору), АТС (номер, район, загальна кількість номерів), <u>Оплата послуг</u> (телефон, АТС, вартість, сплачено чи ні)
10.	Спорт	Спортсмен (ПІБ, рік народження, розряд), Команда (назва, клуб, вид спорту) <u>Нагородження</u> (спортсмен, команда, вид нагороди, дата нагородження)
11.	Господарчі роботи	Підприємство (найменування, розрахунковий рахунок, річний дохід), Земельна ділянка (шифр ділянки, площа, вартість, вид ґрунту), <u>Земле використання</u> (підприємство, земельна ділянка, вид використання: володіння/оренда, термін від, термін до)
12.	Міський транспорт	Транспорт (вид, вартість проїзду, кількість одиниць, рентабельність), Маршрут (початковий пункт, кінцевий пункт, кількість зупинок, довжина, кількість часу в путі), <u>Маршрутний реєстр</u> (транспорт, маршрут, напрямок руху, час виїзду з початкового пункту, час прибуття в кінцевий пункт)

Після цього у відповідній базі даних треба створити таблиці до сутностей власного варіанту, щоб кожна таблиця складалась не менш ніж із десяти записів.

Хід роботи:

1. Створити JSP – сторінку, яка містить меню із посилань для виклику сторінок подальших завдань цієї роботи. Зі сторінок завдань зворотні посилання мають повертати користувача на початкову сторінку із меню. Варіанти із парними номерами формують меню за схемою **JSP – JSP**, непарними – **JSP -Servlet – JSP**. Виконуючи це завдання треба пам'ятати, що сервлет із посилання чи з атрибуту **action** відповідної форми викликається лише за власним паттерном (шаблоном), в той час як JSP може бути викликана тільки з атрибуту action за своїм ім'ям.

Повертатися із сервлета на JSP треба за допомогою об'єкту класу `RequestDispatcher`.

2. За допомогою парсеру DOM для кожної таблиці власної предметної сфери створити відповідний XML-файл, до ім'я якого додати суфікс `dom`, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу. XML-файл до першої таблиці зробити в теговому стилі, до другої – в атрибутному стилі, до третьої – в змішаному стилі (розділ властивостей на атрибути та теги зробити за власним бажанням).
3. *За допомогою парсеру JDOM для кожної таблиці власної предметної сфери створити відповідний XML-файл, до ім'я якого додати суфікс `jdom`, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу. XML-файл до першої таблиці зробити в теговому стилі, до другої – в атрибутному стилі, до третьої – в змішаному стилі (розділ властивостей на атрибути та теги зробити за власним бажанням).
4. *За допомогою парсеру StAX для кожної таблиці власної предметної сфери створити відповідний XML-файл, до ім'я якого додати суфікс `stax`, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу. XML-файл до першої таблиці зробити в теговому стилі, до другої – в атрибутному стилі, до третьої – в змішаному стилі (розділ властивостей на атрибути та теги зробити за власним бажанням).
5. Розробити web-функціонал, який за допомогою парсеру DOM зчитує дані з відповідних XML-файлів (суфікс `dom`) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
6. Розробити web-функціонал, який за допомогою парсеру SAX зчитує дані з відповідних XML-файлів (суфікс `stax`, або `dom`, якщо друге завдання не було виконано) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
7. Розробити web-функціонал, який за допомогою парсеру XPATH зчитує дані з відповідних XML-файлів (суфікс `jdom`, або `dom`, якщо третє завдання не було виконано) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
8. Розробити web-функціонал, який за допомогою front-end технології AJAX зчитує дані з відповідних XML-файлів (будь-який суфікс) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
9. За допомогою бібліотеки JSON.simple для кожної таблиці власної предметної сфери створити відповідний JSON-файл, до ім'я якого додати суфікс `jsons`, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу.

- 10.*За допомогою бібліотеки Jackson для кожної таблиці власної предметної сфери створити відповідний JSON-файл, до ім'я якого додати суфікс jack, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу.
- 11.*За допомогою бібліотеки Gson для кожної таблиці власної предметної сфери створити відповідний JSON-файл, до ім'я якого додати суфікс gson, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу.
- 12.*За допомогою бібліотеки Moshi для кожної таблиці власної предметної сфери створити відповідний JSON-файл, до ім'я якого додати суфікс moshi, і зберегти ці файли у відповідній папці з рекомендацій. Після збереження на відповідній сторінці продемонструвати шлях до створеного файлу.
- 13.Розробити web-функціонал, який за допомогою бібліотеки JSON.simple зчитує дані з відповідних JSON-файлів (суфікс jsons) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
- 14.*Розробити web-функціонал, який за допомогою бібліотеки Jackson зчитує дані з відповідних JSON-файлів (суфікс jack) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
- 15.*Розробити web-функціонал, який за допомогою бібліотеки Gson зчитує дані з відповідних JSON-файлів (суфікс gson) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
- 16.*Розробити web-функціонал, який за допомогою бібліотеки Moshi зчитує дані з відповідних JSON-файлів (суфікс moshi) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).
- 17.Розробити web-функціонал, який за допомогою front-end технології AJAX зчитує дані з відповідних JSON-файлів (будь-який суфікс) та відображає їх у вигляді таблиць на відповідній(их) сторінці(ках).

Лабораторна робота №7

Використання інструменту збірки Gradle під час створення web-застосунків. Створення реєстраційних та авторизаційних функцій у web-системах

Мета: опанувати можливості інструменту збірки Gradle під час розробки реєстраційного та авторизаційного функціоналу умовної webсистеми.

Програмні засоби розробки: середовище програмування (IDE-Integrated Development Environment) IntelliJ IDEA чи Eclipse EE, один чи декілька сучасних web-серверів чи контейнерів: Tomee, Tomcat, Jetty, WildFly (JBoss), GlassFish, бібліотека JSTL.

Рекомендації: після створення проекту лабораторної роботи розгорнути його на усіх наведених вище серверах та контейнерах, що суттєво вплине на кінцевий оціночний бал. Використання скриплетів категорично ЗАБОРОНЕНО. Завдання, помічені *, виконуються на максимальний оціночний бал Сторінки кожного завдання групувати у відповідних папках, а сервлети – в пакетах

Коротка теоретична довідка:

Gradle — це інструмент автоматизації збірки з відкритим кодом, який створюється на основі концепцій Apache Maven і Apache Ant. Він здатний створювати практично будь-яке програмне забезпечення. Він розроблений для створення кількох проектів, які можуть бути досить великими. Він представляє DSL (Domain Specific Language) на основі Java та Groovy замість XML (Extensible Markup Language) для оголошення конфігурації проекту. Він використовує DAG (Directed Acyclic Graph) для визначення порядку виконання завдання. Gradle пропонує еластичну модель, яка може допомогти в життєвому циклі розробки від компіляції та пакування коду для веб- і мобільних програм.

Деякі вагомі переваги Gradle такі:

- Gradle дозволяє писати сценарій збірки за допомогою мови програмування Java;
- простий у використанні та обслуговуванні;
- підтримує керування залежностями;
- забезпечує високу продуктивність і масштабовані збірки;
- процес інтеграції з Gradle набагато простіший;
- підтримує багатопроєктну структуру;
- легко перейти на Gradle з Maven або інших інструментів збірки.

Файл **build.gradle** — це сценарій збірки проекту Gradle. У цьому файлі визначені всі завдання та плагіни.

Типовий файл build.gradle виглядає так:

```
build.gradle
1 /*
2  * This file was generated by the Gradle 'init' task.
3  *
4  * This generated file contains a sample Java Library project to get you started.
5  * For more details take a look at the Java Libraries chapter in the Gradle
6  * User Manual available at https://docs.gradle.org/6.0.1/userguide/java_library_plugin.html
7  */
8
9 plugins {
10     // Apply the java-library plugin to add support for Java Library
11     id 'java-library'
12 }
13
14 repositories {
15     // Use jcenter for resolving dependencies.
16     // You can declare any Maven/Ivy/file repository here.
17     jcenter()
18 }
19
20 dependencies {
21     // This dependency is exported to consumers, that is to say found on their compile classpath.
22     api 'org.apache.commons:commons-math3:3.6.1'
23
24     // This dependency is used internally, and not exposed to consumers on their own compile classpath.
25     implementation 'com.google.guava:guava:28.0-jre'
26
27     // Use JUnit test framework
28     testImplementation 'junit:junit:4.12'
29 }
30
```

Цей файл містить три розділи за замовчуванням:

плагіни: у цьому розділі можна застосувати плагін java-library, щоб додати підтримку бібліотеки java;

репозиторії: у цьому розділі можна оголосити внутрішні та зовнішні репозиторії для вирішення залежностей, оголосити різні типи репозиторіїв, які підтримує Gradle, наприклад Maven, Ant і Ivy;

залежності: у цьому розділі ми можемо оголосити залежності, необхідні для певного проекту.

Крім того, можна оголосити інші пов'язані з проектом модулі як завдання в цьому файлі.

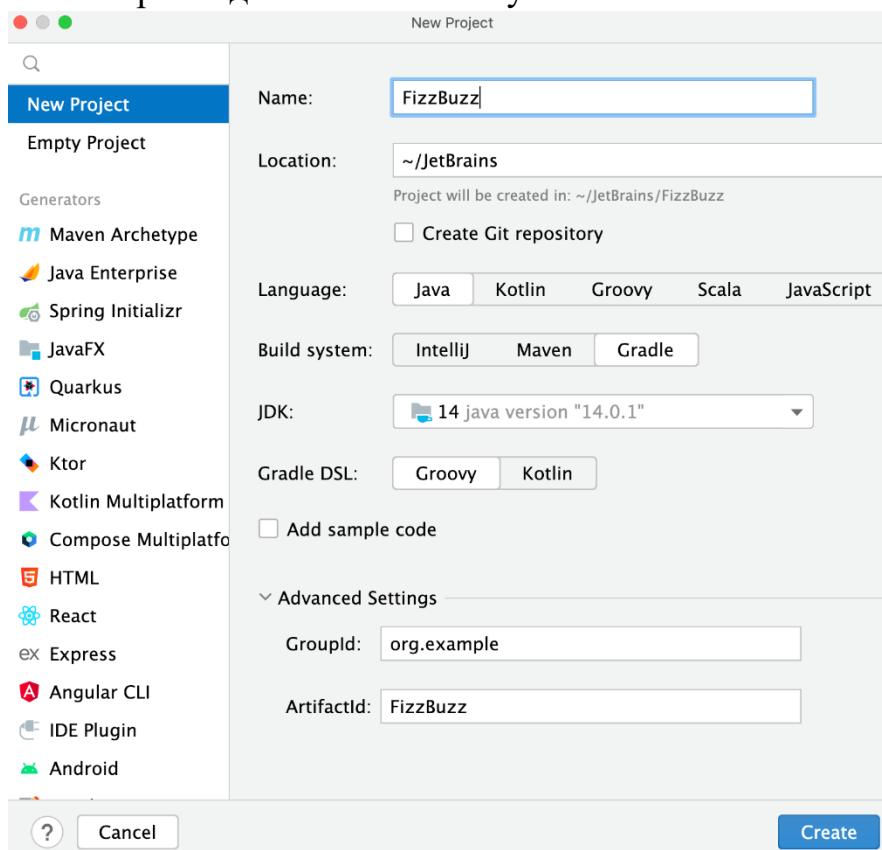
Більшість проектів не є самодостатніми. Їм потрібні деякі файли для компіляції та тестування вихідних файлів. Наприклад, щоб використовувати JSTL, треба включити JSTL JAR (**implementation**

'jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:1.2.7') у шлях до класів.

Gradle використовує певний унікальний сценарій для керування залежностями, який потрібно завантажити.

Залежності використовуються для допомоги у виконанні завдання, наприклад, необхідні файли JAR проекту та зовнішні файли JAR. Кожна залежність застосовується до визначеної області. Наприклад, залежності використовуються для компіляції вихідного коду, і деякі з них будуть доступні під час виконання. Gradle позначає область залежності за допомогою конфігурації, і унікальне ім'я може розпізнати кожну конфігурацію. Більшість плагінів Gradle підтримують попередньо визначену конфігурацію для проектів.

Для використання Gradle-інструментарію під час створення нового проекту треба виконати наприклад ось такі налаштування



Потім додати до файлу build.gradle відповідні залежності до бібліотек, що використовуються в проекті.

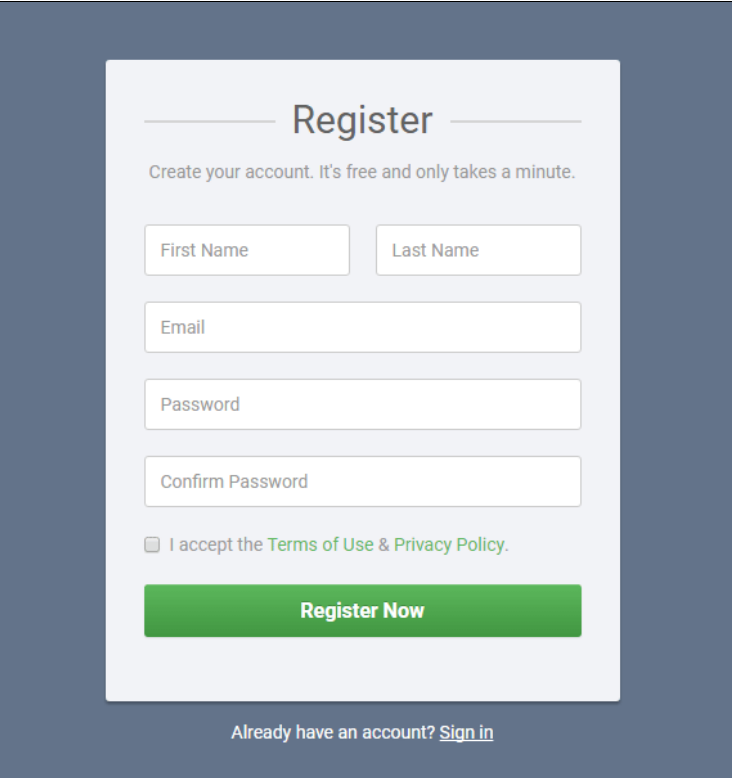
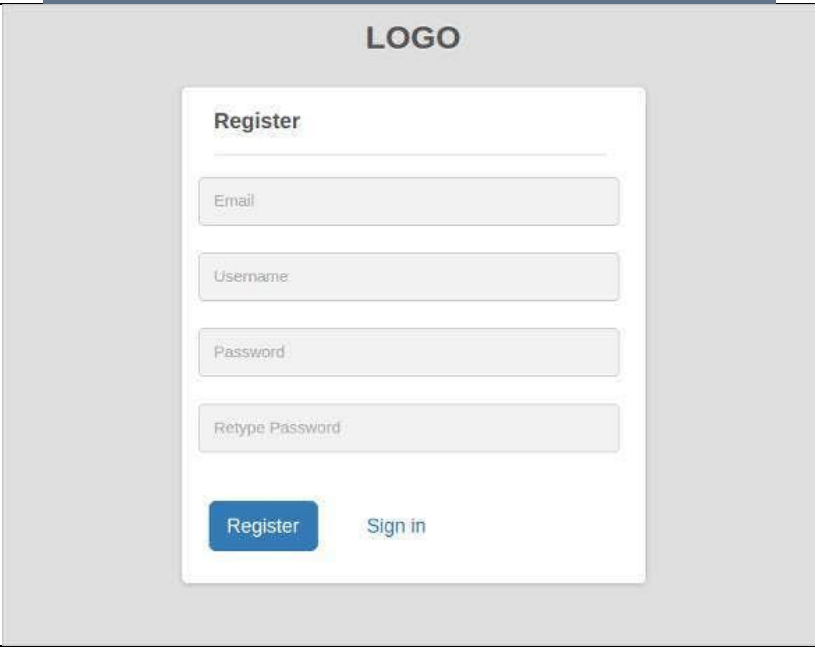
Для детального ознайомлення з особливостями використання інструментарію Gradle можна скористатися посиланням

<https://www.javatpoint.com/gradle-dependencies>

Хід роботи:

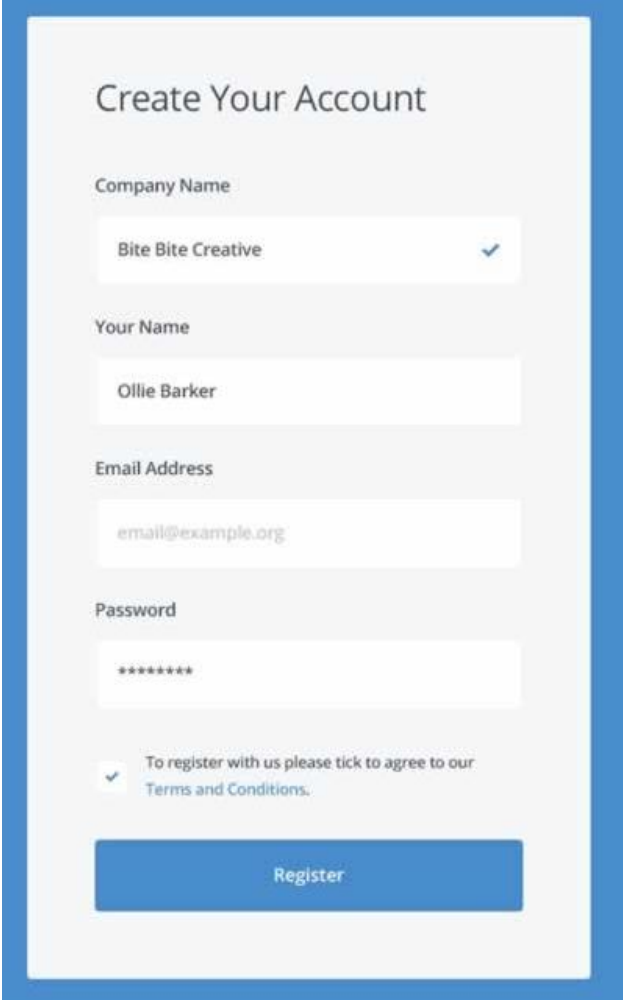
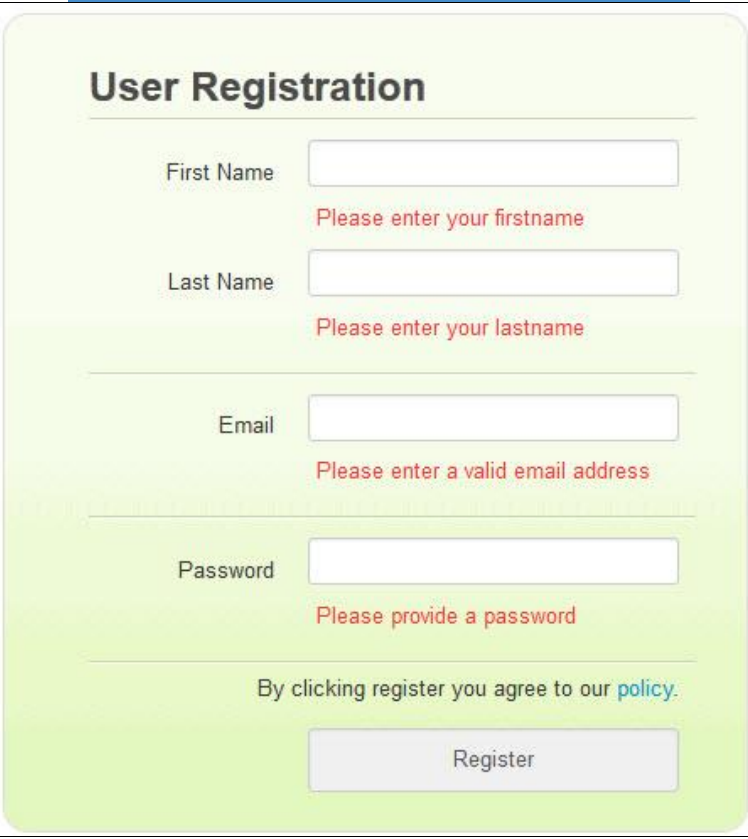
1. Створити початковий Gradle web-проект та перевірити його роботу на одному (бажано на усіх) з найбільш популярних серверів.
2. Додати до проекту функцію реєстрації користувачів умовної web-системи із використанням реєстраційної форми згідно до власного варіанту. Перед занесенням реєстраційних даних до відповідної таблиці, виконати перевірку (валідацію) реєстраційних полів:
 - Username : лише числа та символи латиниці кількістю не більше 15 і щоб ім'я було унікальним (раніше не було в таблиці БД);
 - Password : числа, символи латиниці та обов'язково спеціальні символи довжиною не менше 12, та його коректне повторення у відповідному полі;
 - Email : коректне значення електронної поштової адреси





В разі наявності помилок повертатися до форми і відображати їх навпроти відповідних полів

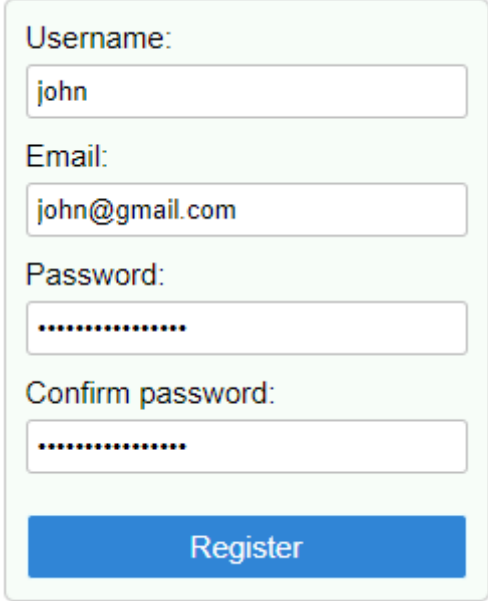
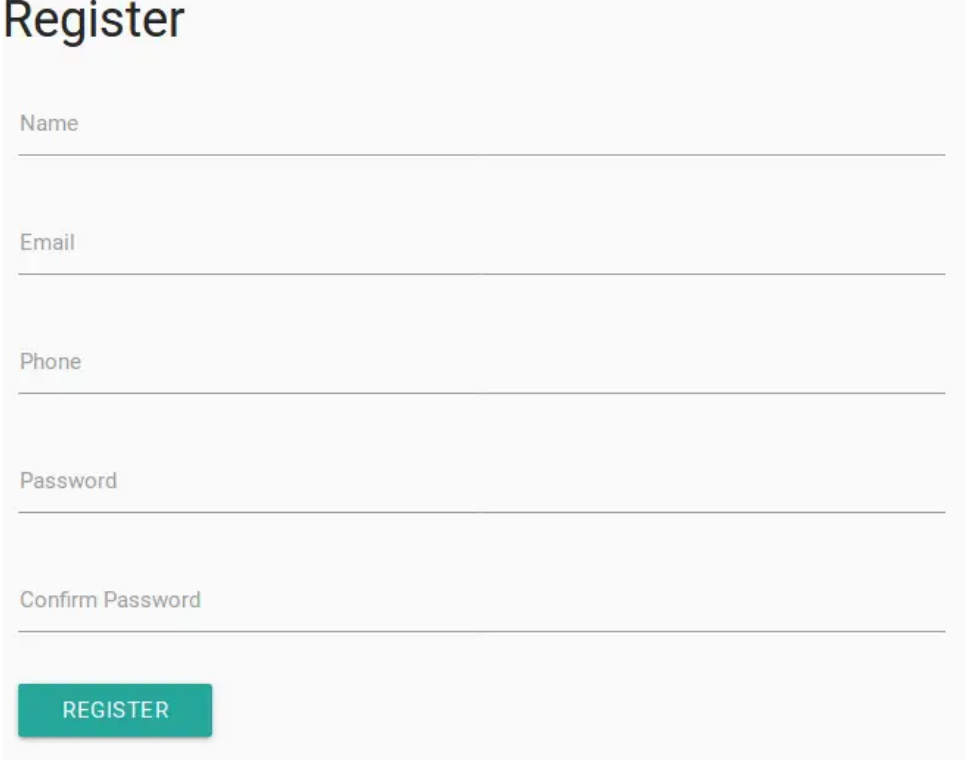
№ варі- анту	Шаблон реєстраційної форми	
1	2	
1.	 A registration form titled "Register" with a subtitle "Create your account. It's free and only takes a minute." The form includes input fields for "First Name", "Last Name", "Email", "Password", and "Confirm Password". Below these fields is a checkbox labeled "I accept the Terms of Use & Privacy Policy." and a green "Register Now" button. At the bottom, there is a link "Already have an account? Sign in".	
2.	 A registration form titled "Register" with a "LOGO" placeholder at the top. The form includes input fields for "Email", "Username", "Password", and "Retype Password". At the bottom, there are two buttons: a blue "Register" button and a "Sign in" link.	

1	2
3.	<div data-bbox="478 210 1321 909"><h2>Register</h2><p>Kindly fill in this form to register.</p><p>Username</p><input data-bbox="643 454 1158 490" type="text"/><p>Email</p><input data-bbox="643 544 1158 580" type="text"/><p>Password</p><input data-bbox="643 633 1158 669" type="text"/><p>Repeat Password</p><input data-bbox="643 723 1158 759" type="text"/><p><input data-bbox="643 781 1158 817" type="button" value="Register"/></p><p>Already have an account? Log in.</p></div>
4.	<div data-bbox="560 936 1254 1856"><div data-bbox="901 954 1230 1032">Login×</div><div data-bbox="584 1032 1222 1845"><h3>New account</h3><p>User Name:</p><input data-bbox="632 1171 1177 1249" type="text"/><p>Password:</p><input data-bbox="632 1301 1177 1379" type="text"/><p>Email:</p><input data-bbox="632 1431 1177 1509" type="text"/><p>Birth date</p><p>Year: <input data-bbox="632 1637 746 1715" type="text"/> Month: <input data-bbox="767 1637 839 1715" type="text"/> Day: <input data-bbox="860 1637 932 1715" type="text"/></p><p><input data-bbox="798 1771 1011 1845" type="button" value="CREATE"/></p></div></div>

1	2
5.	<div data-bbox="497 197 1319 1285"><div><h3>Register</h3><p>Insert text here.</p><p>Error Error Error</p><div><input type="text" value="Name"/><input type="password" value="E-mail Address"/><input type="password" value="Password"/><input type="password" value="Confirm Password"/><div><div>Register</div><div>Clear Form</div></div></div><p>Already have an account? Login here.</p></div></div>
6.	<div data-bbox="456 1285 1361 2045"><div>Registration Example JavaFX</div><div><div>Your Name</div><input type="text"/><div>Your Username</div><input type="text"/><div>Your Password</div><input type="password"/><div>Confirm Password</div><input type="password"/><div>REGISTER</div></div></div>

1	2
7.	 <p>The image shows a 'Create Your Account' form with a blue border. It includes fields for Company Name (filled with 'Bite Bite Creative'), Your Name (filled with 'Ollie Barker'), Email Address (filled with 'email@example.org'), and Password (filled with '*****'). There is a checkbox for 'To register with us please tick to agree to our Terms and Conditions.' which is checked. A blue 'Register' button is at the bottom.</p>
8.	 <p>The image shows a 'User Registration' form with a light green background. It includes fields for First Name, Last Name, Email, and Password, each with a red error message below it: 'Please enter your firstname', 'Please enter your lastname', 'Please enter a valid email address', and 'Please provide a password'. There is a checkbox for 'By clicking register you agree to our policy.' which is checked. A grey 'Register' button is at the bottom.</p>

1	2
9.	<div><h1>Register</h1><div><div>Username</div><div>Enter username</div></div><div><div>Enter Password</div><div>EnterPassword</div></div><div><div>Confirm Password</div><div>Confirm Password</div></div><div>By creating an account you agree to our Terms & Privacy.</div><div>Register</div></div>
10.	<div><h1>SIGN UP</h1><div><div>Your username</div><div> mysuperusername690</div></div><div><div>Your email</div><div> mysupermail@mail.com</div></div><div><div>Your password</div><div> eg. X8df#90EO</div></div><div><div>Please confirm your password</div><div> eg. X8df#90EO</div></div><div>SIGN UP</div><div>Already a member ? Go and log in</div></div>

1	2
11.	
12.	

Валідацію полів форми можна робити класичними засобами технології front-end

3. Додати до проекту функцію авторизації користувачів умовної web-системи із використанням дизайну реєстраційної форми з минулого завдання відповідного варіанту. Авторизаційна форма має складатися з двох полів: Username та Password, до яких користувач має ввести відповідну інформацію. У випадку коректного вводу система має підтвердити авторизацію користувача, у протилежному випадку – повернутися до назад форми та видати повідомлення про помилку в позиції згідно до свого варіанта

№ варіанту	Відносна локація (сірий відтінок) повідомлень про помилки			
1	2			
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				

1	2			
9.				
10.				
11.				
12.				

4. Об'єднати функції реєстрації та авторизації та зробити загальний вхідний функціонал умовної web-системи. Перевірити коректність його роботи.