**Лабораторна робота №9**

**Тема**: **ООП в JavaScript.**

**Мета роботи:** вивчити особливості використання ООП в JavaScript.

**Хід роботи:**

**Завдання 1.**

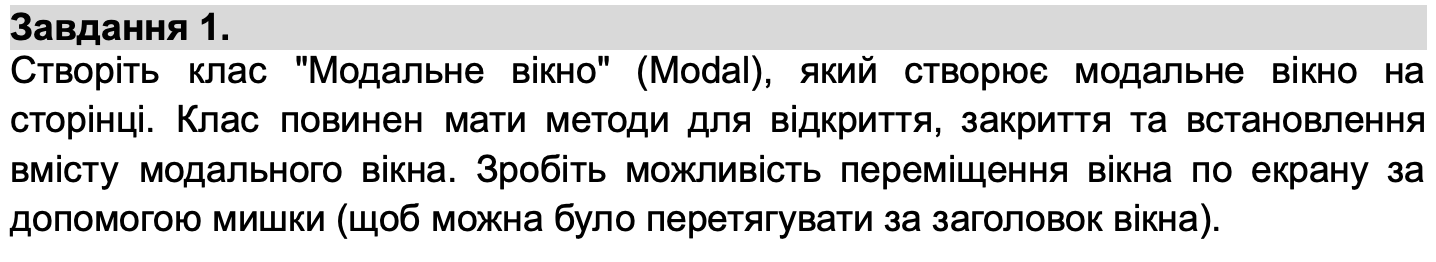


Рис. 1. Завдання 1

Виконання завдання:

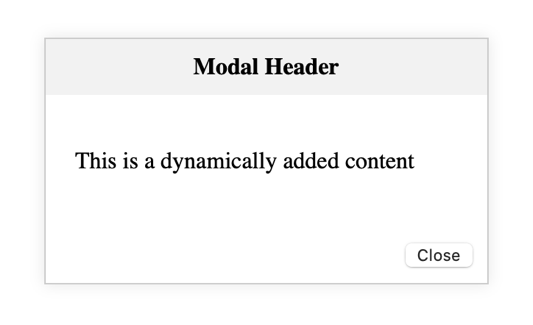
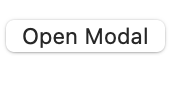


Рис. 2-3. Виконання завдання 1

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 <title>Task 1</title>  
 </head>  
 <body>  
 <button id="open-modal">Open Modal</button>  
 <div id="modal" class="modal hidden">  
 <div id="modal-header" class="modal-header">Modal Header</div>

<div id="modal-content" class="modal-content">Modal Content</div>  
 <button id="modal-close" class="modal-close">Close</button>  
 </div>  
 </body>  
</html>

Css:

.modal {  
 position: fixed;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 width: 300px;  
 background-color: white;  
 border: 1px solid #ccc;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 display: flex;  
 flex-direction: column;  
 z-index: 1000;  
 opacity: 0;  
 pointer-events: none;  
 transition: opacity 0.3s ease, transform 0.3s ease;  
}  
  
.modal:not(.hidden) {  
 opacity: 1;  
 pointer-events: auto;  
}  
  
.hidden {  
 display: none;  
}  
  
.modal-header {  
 padding: 10px;  
 background-color: #f2f2f2;  
 cursor: move;  
 font-weight: bold;  
 text-align: center;  
}  
  
.modal-content {  
 padding: 20px;  
}  
  
.modal-close {  
 align-self: flex-end;  
 margin: 10px;  
}  
  
.modal-backdrop {  
 position: fixed;  
 top: 0;  
 left: 0;

width: 100%;  
 height: 100%;  
 background-color: rgba(0, 0, 0, 0.5);  
 z-index: 999;  
 opacity: 0;  
 pointer-events: none;  
 transition: opacity 0.3s ease;  
}  
  
.modal-backdrop:not(.hidden) {  
 opacity: 1;  
 pointer-events: auto;  
}

Js:

class Modal {  
 constructor() {  
 this.modal = *document*.getElementById("modal");  
 this.header = *document*.getElementById("modal-header");  
 this.closeButton = *document*.getElementById("modal-close");  
 this.content = *document*.getElementById("modal-content");  
  
 // Динамічний бекдроп  
 this.backdrop = *document*.createElement("div");  
 this.backdrop.className = "modal-backdrop hidden";  
 *document*.body.appendChild(this.backdrop);  
  
 this.offsetX = 0;  
 this.offsetY = 0;  
 this.isDragging = false;  
  
 this.initEvents();  
 }  
  
 open() {  
 this.modal.classList.remove("hidden");  
 this.backdrop.classList.remove("hidden");  
 }  
  
 close() {  
 this.modal.classList.add("hidden");  
 this.backdrop.classList.add("hidden");  
 }  
  
 setContent(content) {  
 if (this.content) {  
 this.content.innerHTML = content;  
 } else {  
 *console*.error("Modal content element not found.");  
 }  
 }  
  
 initEvents() {  
 this.closeButton.addEventListener("click", this.close.bind(this));  
 this.backdrop.addEventListener("click", this.close.bind(this));

this.header.addEventListener("mousedown", this.startDrag.bind(this));  
 *document*.addEventListener("mousemove", this.onDrag.bind(this));  
 *document*.addEventListener("mouseup", this.stopDrag.bind(this));  
 }  
  
 startDrag(event) {  
 this.isDragging = true;  
 this.offsetX = event.clientX - this.modal.offsetLeft;  
 this.offsetY = event.clientY - this.modal.offsetTop;  
 }  
  
 onDrag(event) {  
 if (this.isDragging) {  
 this.modal.style.left = `${event.clientX - this.offsetX}px`;  
 this.modal.style.top = `${event.clientY - this.offsetY}px`;  
 }  
 }  
  
 stopDrag() {  
 this.isDragging = false;  
 }  
}  
  
// Використання  
const *modal* = new Modal();  
*document*.getElementById("open-modal").addEventListener("click", () => {  
 *modal*.setContent("<p>This is a dynamically added content</p>");  
 *modal*.open();  
});

**Завдання 2.**

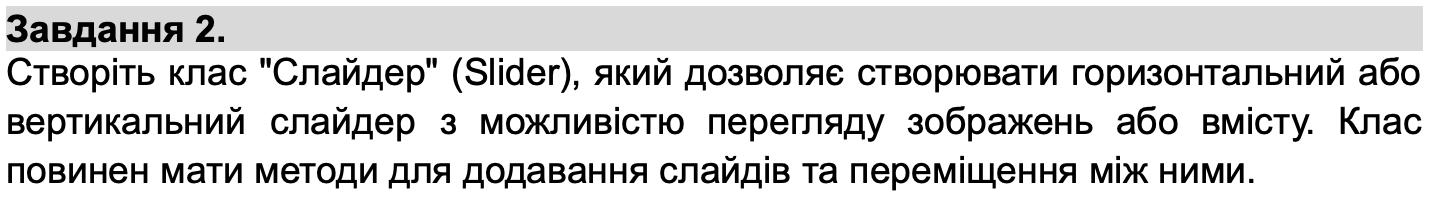


Рис. 4. Завдання 2

Виконання завдання:



Рис. 5. Виконання завдання 2

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">

<head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 <title>Task 2</title>  
 </head>  
 <body>  
 <div id="slider-container" class="slider-container">  
 <button id="prev-slide" class="slider-button">Prev</button>  
 <div id="slider" class="slider"></div>  
 <button id="next-slide" class="slider-button">Next</button>  
 </div>  
 </body>  
</html>

Css:

.slider-container {  
 display: flex;  
 align-items: center;  
 overflow: hidden;  
 width: 300px;  
 height: 200px;  
 position: relative;  
}  
.slider {  
 display: flex;  
 transition: transform 0.5s ease;  
 flex-direction: row;  
 width: 100%;  
}  
.slider-button {  
 background-color: #333;  
 color: white;  
 border: none;  
 padding: 10px;  
 cursor: pointer;  
 position: absolute;  
 top: 50%;  
 transform: translateY(-50%);  
 z-index: 1;  
}  
#prev-slide {  
 left: 10px;  
}  
  
#next-slide {  
 right: 10px;  
}  
.slider-item {  
 min-width: 100%;  
 height: 100%;  
 display: flex;  
 align-items: center;  
 justify-content: center;

font-size: 24px;  
 color: white;  
 background-color: #555;  
}

Js:

class Slider {  
 constructor(containerId, orientation = 'horizontal') {  
 this.container = *document*.getElementById(containerId);  
 this.slider = this.container.querySelector('.slider');  
 this.slides = [];  
 this.currentIndex = 0;  
 this.orientation = orientation;  
  
 this.initEvents();  
 this.updateSlidePosition();  
 }  
  
 // Додає новий слайд до слайдера  
 addSlide(content) {  
 const slide = *document*.createElement('div');  
 slide.classList.add('slider-item');  
 slide.innerHTML = content;  
 this.slider.appendChild(slide);  
 this.slides.push(slide);  
 this.updateSlidePosition();  
 }  
  
 // Переміщення до наступного слайда  
 nextSlide() {  
 if (this.currentIndex < this.slides.length - 1) {  
 this.currentIndex++;  
 } else {  
 this.currentIndex = 0;  
 }  
 this.updateSlidePosition();  
 }  
  
 // Переміщення до попереднього слайда  
 prevSlide() {  
 if (this.currentIndex > 0) {  
 this.currentIndex--;  
 } else {  
 this.currentIndex = this.slides.length - 1;  
 }  
 this.updateSlidePosition();  
 }  
  
 // Оновлює позицію слайдів відповідно до поточного індексу  
 updateSlidePosition() {  
 const offset = this.orientation === 'horizontal'  
 ? -this.currentIndex \* 100  
 : -this.currentIndex \* 100;

if (this.orientation === 'horizontal') {  
 this.slider.style.transform = `translateX(${offset}%)`;  
 } else {  
 this.slider.style.flexDirection = 'column';  
 this.slider.style.transform = `translateY(${offset}%)`;  
 }  
 }  
  
 // Ініціалізація подій для кнопок  
 initEvents() {  
 const nextButton = this.container.querySelector('#next-slide');  
 const prevButton = this.container.querySelector('#prev-slide');  
  
 nextButton.addEventListener('click', () => this.nextSlide());  
 prevButton.addEventListener('click', () => this.prevSlide());  
 }  
}  
  
const *slider* = new Slider('slider-container', 'horizontal');  
*slider*.addSlide('<div>Slide 1</div>');  
*slider*.addSlide('<div>Slide 2</div>');  
*slider*.addSlide('<div>Slide 3</div>');

**Завдання 3.**

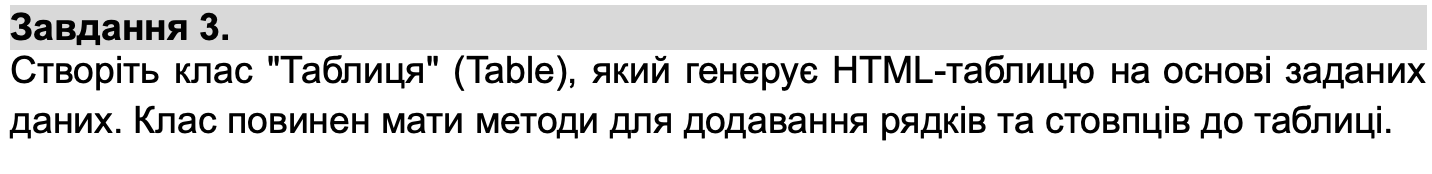


Рис. 6. Завдання 3

Виконання завдання:

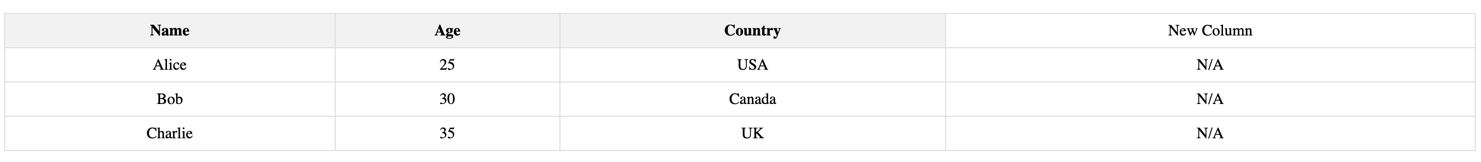


Рис. 7. Виконання завдання 3

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 <title>Task 3</title>  
 </head>  
 <body>

<div id="table-container"></div>  
 </body>  
</html>

Css:

table {  
 width: 100%;  
 border-collapse: collapse;  
 margin-top: 20px;  
}  
  
th, td {  
 border: 1px solid #ddd;  
 padding: 8px;  
 text-align: center;  
}  
  
th {  
 background-color: #f2f2f2;  
 font-weight: bold;  
}

Js:

class Table {  
 constructor(containerId) {  
 this.container = *document*.getElementById(containerId);  
 this.table = *document*.createElement('table');  
 this.container.appendChild(this.table);  
 }  
  
 // Метод для додавання заголовків стовпців  
 setHeaders(headers) {  
 const thead = this.table.createTHead();  
 const row = thead.insertRow();  
 headers.forEach(headerText => {  
 const th = *document*.createElement('th');  
 th.textContent = headerText;  
 row.appendChild(th);  
 });  
 }  
  
 // Метод для додавання рядка до таблиці  
 addRow(data) {  
 const row = this.table.insertRow();  
 data.forEach(cellData => {  
 const cell = row.insertCell();  
 cell.textContent = cellData;  
 });  
 }  
  
 // Метод для додавання стовпця до існуючих рядків  
 addColumn(defaultValue = '') {  
 const rows = this.table.rows;

*Array*.from(rows).forEach((row, index) => {  
 const cell = row.insertCell();  
 cell.textContent = index === 0 ? "New Column" : defaultValue;  
 });  
 }  
}  
  
// Використання класу  
const *table* = new Table('table-container');  
  
// Додаємо заголовки стовпців  
*table*.setHeaders(['Name', 'Age', 'Country']);  
  
// Додаємо кілька рядків  
*table*.addRow(['Alice', 25, 'USA']);  
*table*.addRow(['Bob', 30, 'Canada']);  
*table*.addRow(['Charlie', 35, 'UK']);  
  
// Додаємо новий стовпець з дефолтним значенням  
*table*.addColumn('N/A');

**Завдання 4.**

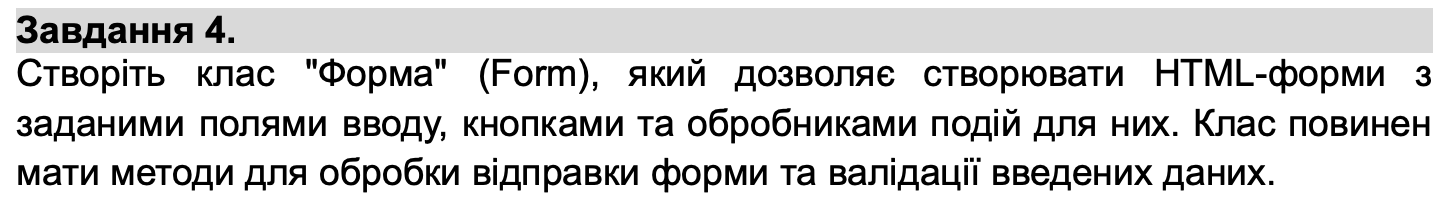


Рис. 8. Завдання 4

Виконання завдання:

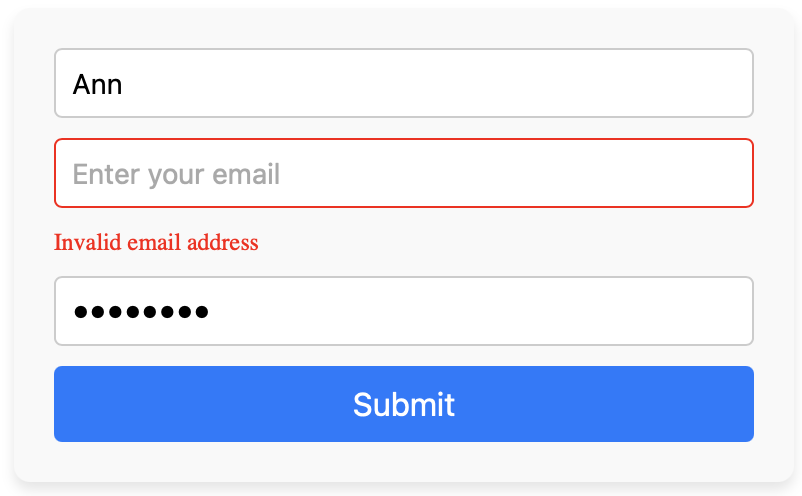
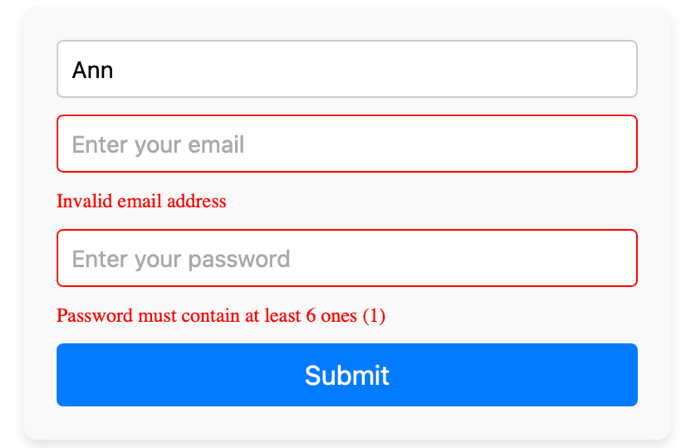


Рис. 9-10. Виконання завдання 4

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">  
 <head>

<meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Task 4</title>  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 </head>  
 <body>  
 <div id="form-container"></div>  
 </body>  
</html>

Css:

form {  
 display: flex;  
 flex-direction: column;  
 width: 350px;  
 margin: 20px auto;  
 background-color: #f9f9f9;  
 padding: 20px;  
 border-radius: 8px;  
 box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  
}  
input {  
 margin-bottom: 10px;  
 padding: 8px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 font-size: 14px;  
}  
button {  
 padding: 10px;  
 font-size: 16px;  
 color: white;  
 background-color: #007BFF;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
}  
button:hover {  
 background-color: #0056b3;  
}  
  
input.invalid {  
 border: 1px solid red;  
}  
  
.error-message {  
 color: red;  
 font-size: 12px;  
 margin-bottom: 10px;  
}

Js:

class Form {  
 constructor(containerId, fields, onSubmit) {  
 // Ідентифікатор контейнера, куди додається форма  
 this.container = *document*.getElementById(containerId);  
  
 // Поля форми (масив об'єктів з описом кожного поля)  
 this.fields = fields;  
  
 // Функція-обробник для відправки форми  
 this.onSubmit = onSubmit;  
  
 // Створюємо HTML-елемент форми  
 this.form = *document*.createElement('form');  
 this.container.appendChild(this.form);  
  
 // Рендеримо поля форми  
 this.renderFields();  
  
 // Додаємо кнопку "Submit"  
 this.addSubmitButton();  
 }  
  
 // Метод для створення та відображення полів форми  
 renderFields() {  
 let self = this; // Зберігаємо посилання на поточний об'єкт  
 this.fields.forEach(function (field) {  
 let input = *document*.createElement('input');  
 input.type = field.type || 'text'; // Тип поля (текст за замовчуванням)  
 input.name = field.name; // Ім'я поля  
 input.placeholder = field.placeholder || ''; // Підказка в полі  
 input.required = field.required || false; // Чи є поле обов'язковим  
 input.classList.add(field.name); // Додаємо клас за назвою поля  
  
 // Додаємо поле до форми  
 self.form.appendChild(input);  
 });  
 }  
  
 // Метод для створення кнопки відправки форми  
 addSubmitButton() {  
 let button = *document*.createElement('button');  
 button.type = 'button'; // Тип кнопки — просто виконує дію без перезавантаження  
 button.textContent = 'Submit'; // Текст кнопки  
 this.form.appendChild(button);  
  
 // Додаємо обробник події для натискання на кнопку  
 button.addEventListener('click', this.handleSubmit.bind(this)); // Прив'язуємо контекст  
 }  
  
 // Метод для обробки відправки форми  
 handleSubmit(event) {  
 let self = this; // Зберігаємо посилання на поточний об'єкт  
 let isValid = true; // Прапорець для перевірки валідності всіх полів  
 let data = {}; // Об'єкт для збереження даних форми

// Перевіряємо кожне поле на валідність  
 this.fields.forEach(function (field) {  
 let input = self.form.querySelector('.' + field.name);  
 let errorMessageElement = input.nextElementSibling;  
  
 // Якщо помилка вже існує, видаляємо її  
 if (errorMessageElement && errorMessageElement.classList.contains('error-message')) {  
 errorMessageElement.remove();  
 }  
  
 // Якщо поле не пройшло валідацію  
 if (field.validation && !field.validation(input.value)) {  
 isValid = false;  
  
 // Додаємо клас для позначення некоректного поля  
 input.classList.add('invalid');  
  
 // Створюємо повідомлення про помилку  
 let errorMessage = *document*.createElement('div');  
 errorMessage.classList.add('error-message');  
 errorMessage.textContent = field.errorMessage || 'Invalid input';  
  
 // Додаємо повідомлення після поля  
 input.after(errorMessage);  
 } else {  
 // Якщо поле валідне, прибираємо некоректний клас  
 input.classList.remove('invalid');  
 // Зберігаємо дані в об'єкт  
 data[field.name] = input.value;  
 }  
 });  
  
 // Якщо всі поля валідні, викликаємо onSubmit  
 if (isValid) {  
 this.onSubmit(data); // Викликаємо передану функцію  
 this.form.reset(); // Очищуємо форму після успішної відправки  
 }  
 }  
}  
  
// Використання класу  
let *form* = new Form('form-container', [  
 {  
 name: 'username',  
 placeholder: 'Enter your username',  
 required: true,  
 validation: function (value) {  
 return value.length >= 3; // Ім'я повинно бути щонайменше 3 символи  
 },  
 errorMessage: 'Username must be at least 3 characters long'  
 },  
 {  
 name: 'email',  
 placeholder: 'Enter your email',  
 type: 'email',  
 required: true,  
 validation: function (value) {

return /^\S+@\S+\.\S+$/.test(value); // Перевірка формату email  
 },  
 errorMessage: 'Invalid email address'  
 },  
 {  
 name: 'password',  
 placeholder: 'Enter your password',  
 type: 'password',  
 required: true,  
 validation: function (value) {  
 return (value.match(/1/g) || []).length >= 6; // Пароль повинен містити щонайменше 6 одиниць  
 },  
 errorMessage: 'Password must contain at least 6 ones (1)'  
 }  
], function (data) {  
 *console*.log('Form submitted:', data); // Вивід даних у консоль  
});

**Завдання 5.**

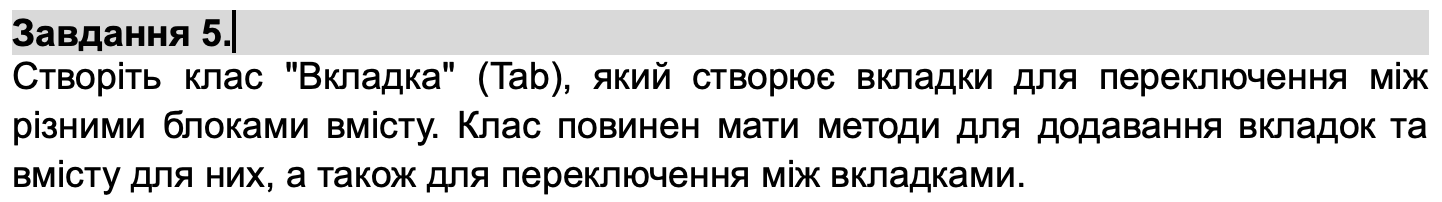


Рис. 11. Завдання 5

Виконання завдання:

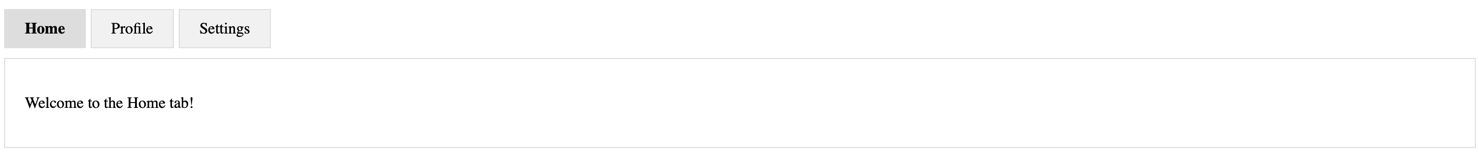


Рис. 12. Виконання завдання 5

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 <title>Task 5</title>  
 </head>  
 <body>  
 <div id="tab-container"></div>  
 </body>  
</html>

Css:

.tabs {  
 display: flex;  
 cursor: pointer;  
 margin-bottom: 10px;  
}  
.tab {  
 padding: 10px 20px;  
 border: 1px solid #ddd;  
 background-color: #f1f1f1;  
 margin-right: 5px;  
 user-select: none;  
}  
.tab.active {  
 background-color: #ddd;  
 font-weight: bold;  
}  
.content {  
 display: none;  
 padding: 20px;  
 border: 1px solid #ddd;  
}  
.content.active {  
 display: block;  
}

Js:

class Tab {  
 constructor(containerId) {  
 // Знаходимо контейнер для вкладок  
 this.container = *document*.getElementById(containerId);  
  
 // Створюємо елемент заголовків вкладок  
 this.tabHeaders = *document*.createElement('div');  
 this.tabHeaders.classList.add('tabs');  
 this.container.appendChild(this.tabHeaders);  
  
 // Створюємо елемент для вмісту вкладок  
 this.contents = *document*.createElement('div');  
 this.contents.classList.add('contents');  
 this.container.appendChild(this.contents);  
  
 // Масиви для збереження вкладок і їхнього вмісту  
 this.tabs = [];  
 this.contentsList = [];  
 }  
  
 // Метод для додавання вкладки  
 addTab(title, content) {  
 let tabIndex = this.tabs.length;  
  
 // Створюємо заголовок вкладки  
 let tab = *document*.createElement('div');  
 tab.classList.add('tab');

tab.textContent = title;  
  
 // Додаємо обробник події для перемикання  
tab.addEventListener('click', this.switchTab.bind(this, tabIndex));  
  
 this.tabHeaders.appendChild(tab);  
 this.tabs.push(tab);  
  
 // Створюємо блок вмісту вкладки  
 let contentDiv = *document*.createElement('div');  
 contentDiv.classList.add('content');  
 contentDiv.innerHTML = content;  
  
 this.contents.appendChild(contentDiv);  
 this.contentsList.push(contentDiv);  
  
 // Активуємо першу вкладку за замовчуванням  
 if (this.tabs.length === 1) {  
 this.activateTab(0);  
 }  
 }  
  
 // Метод для перемикання між вкладками  
 switchTab(index) {  
 let self = this; // Зберігаємо контекст  
 this.tabs.forEach(function (tab, i) {  
 if (i === index) {  
 tab.classList.add('active');  
 self.contentsList[i].classList.add('active');  
 } else {  
 tab.classList.remove('active');  
 self.contentsList[i].classList.remove('active');  
 }  
 });  
 }  
  
 // Метод для активації вкладки  
 activateTab(index) {  
 this.switchTab(index);  
 }  
}  
  
// Використання класу  
let *tabComponent* = new Tab('tab-container');  
*tabComponent*.addTab('Home', '<p>Welcome to the Home tab!</p>');  
*tabComponent*.addTab('Profile', '<p>This is your Profile.</p>');  
*tabComponent*.addTab('Settings', '<p>Here you can change your settings.</p>');

**Завдання 6.**

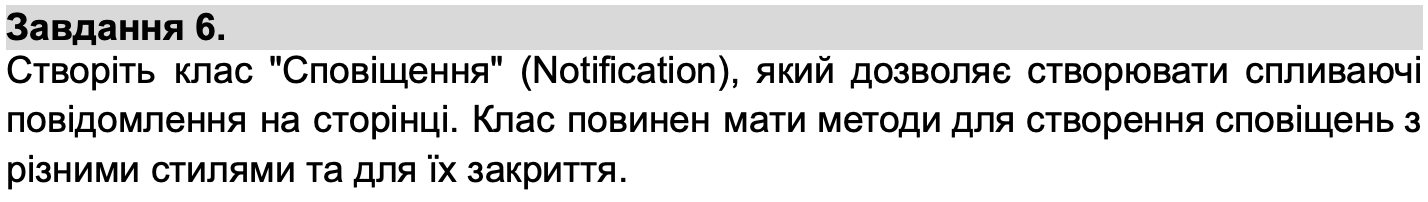


Рис. 13. Завдання 6

Виконання завдання:

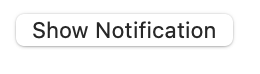


Рис. 14. Завдання 6

Програма до виконаного завдання:

Html:

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="style.css">  
 <script defer src="script.js"></script>  
 <title>Task 6</title>  
 </head>  
 <body>  
 <button onclick="showNotification()">Show Notification</button>  
 <div id="notifications-container"></div> <!-- Контейнер для сповіщень -->  
 </body>  
</html>

Css:

/\* Контейнер для повідомлень \*/  
#notifications-container {  
 position: fixed;  
 top: 10px;  
 right: 10px;  
 z-index: 1000;  
}  
  
/\* Стиль для кожного повідомлення \*/  
.notification {  
 position: relative;  
 background-color: #f0f0f0;  
 color: #333;  
 padding: 10px 40px 10px 10px; /\* Відступи враховують кнопку закриття \*/  
 margin-bottom: 10px;  
 border-radius: 5px;  
 box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  
 transition: opacity 0.5s ease-in-out;  
}  
  
/\* Стиль для успішного повідомлення \*/  
.notification.success {

background-color: #4CAF50;  
 color: white;  
}  
  
/\* Стиль для попереджень \*/  
.notification.warning {  
 background-color: #FFC107;  
 color: white;  
}  
  
/\* Стиль для помилок \*/  
.notification.error {  
 background-color: #F44336;  
 color: white;  
}  
  
/\* Стиль для кнопки закриття \*/  
.notification .close-btn {  
 position: absolute;  
 top: 5px;  
 right: 10px;  
 background: none;  
 border: none;  
 color: white;  
 font-size: 18px;  
 cursor: pointer;  
}  
  
/\* Анімація для приховування повідомлення \*/  
.notification.hidden {  
 opacity: 0;  
 pointer-events: none;  
}

Js:

class Notification {  
 constructor(containerId) {  
 // Ініціалізація контейнера для сповіщень  
 this.container = *document*.getElementById(containerId);  
 }  
  
 // Метод для створення сповіщення  
 createNotification(message, type = 'info') {  
 // Створення контейнера для сповіщення  
 let notification = *document*.createElement('div');  
 notification.classList.add('notification', type);  
  
 // Створення кнопки закриття  
 let closeButton = *document*.createElement('button');  
 closeButton.textContent = '×';  
 closeButton.classList.add('close-btn');  
 closeButton.addEventListener('click', this.closeNotification.bind(this, notification));  
  
 // Додавання кнопки та тексту до сповіщення  
 notification.appendChild(closeButton);

let messageSpan = *document*.createElement('span');  
 messageSpan.textContent = message;  
 notification.appendChild(messageSpan);  
  
 // Додавання сповіщення в контейнер  
 this.container.appendChild(notification);  
  
 // Автоматичне приховування сповіщення через 5 секунд  
 setTimeout(() => {  
 this.closeNotification(notification);  
 }, 5000);  
 }  
  
 // Метод для закриття сповіщення  
 closeNotification(notification) {  
 notification.classList.add('hidden');  
 setTimeout(() => {  
 if (notification.parentNode) {  
 notification.remove();  
 }  
 }, 500); // Затримка перед видаленням з DOM  
 }  
}  
  
// Ініціалізація менеджера сповіщень  
let *notificationManager* = new Notification('notifications-container');  
  
// Функція для створення сповіщень  
function showNotification() {  
 *notificationManager*.createNotification('This is an info message.', 'info');  
 *notificationManager*.createNotification('This is a success message!', 'success');  
 *notificationManager*.createNotification('Warning! Something went wrong.', 'warning');  
 *notificationManager*.createNotification('Error! Something failed.', 'error');  
}

**Висновок:** У ході виконання лабораторної роботи №9 я вивчила основи об'єктно-орієнтованого програмування (ООП) в JavaScript. Було досліджено ключові концепції, такі як створення класів, використання конструкторів, методів та властивостей. Також ознайомилася з можливостями наслідування, поліморфізму та інкапсуляції. Результатом роботи стало закріплення практичних навичок створення об'єктів і взаємодії між ними, що дозволяє ефективніше будувати складні та масштабовані програми.