Mobile Computing : Lecture-5

Created by	Ahmed Bashir
Email	ahmed.bashir@gift.edu.pk

Mobile Computing : Lecture-5

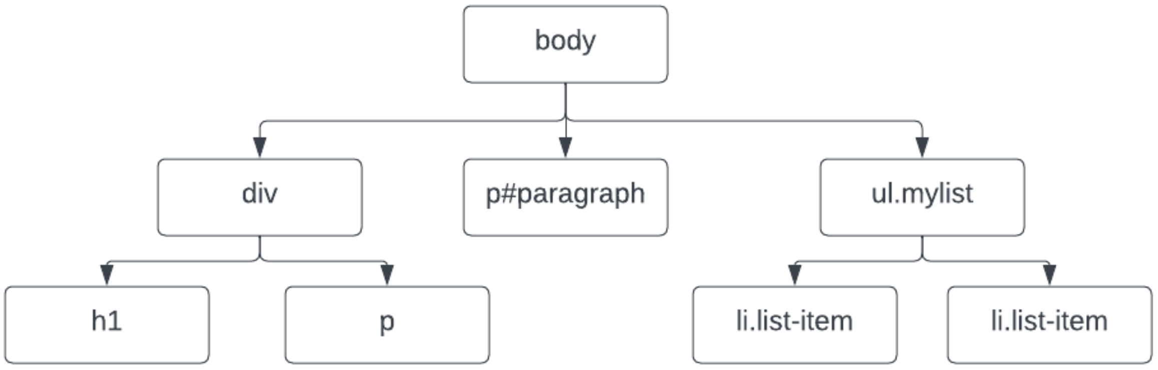
DOM (Document Object Model)

DOM stands for Document Object Model. It is a standard object-oriented interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document. The DOM represents a document as a tree structure, with each node in the tree representing a different part of the document.

DOM manipulation involves accessing and modifying HTML elements and their attributes using JavaScript. There are several ways to manipulate the DOM in JavaScript.

Consider the following HTML snippet

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Document</title>
</head>
<body>
    <div>
        <h1>Lecture-5</h1>
        <p>This lecture is about DOM</p>
    </div>
    <p id="paragraph">A paragraph with id</p>
    <ul class="mylist">
        <li class="list-item">Item 1</li>
        <li class="list-item">Item 2</li>
    </ul>
</body>
</html>
```



A DOM Tree representing the above given HTML code

Note that `#` is used to represent an `id` and `.` (dot) is used to represent a `class`.

Commonly Used DOM Manipulation Methods

Here is a table summarizing the commonly used DOM manipulation methods.

Method	Description
`document.getElementById(id)`	Returns the element with the specified ID
`document.querySelector(selector)`	Returns the first element that matches the specified CSS selector
`document.createElement(tagName)`	Creates a new HTML element with the specified tag name
`element.appendChild(newChild)`	Adds a new child node to the end of the specified element
`element.removeChild(child)`	Removes a child node from the specified element
`element.setAttribute(name, value)`	Sets the value of the specified attribute for the element
`element.getAttribute(name)`	Returns the value of the specified attribute for the element
`element.innerHTML`	Sets or returns the HTML content of the specified element
`element.style.property`	Sets or returns the value of the specified CSS

	property for the element
`element.classList.add(className)`	Adds a class to the element's list of classes
`element.classList.remove(className)`	Removes a class from the element's list of classes
`element.classList.toggle(className)`	Toggles a class on or off for the element
`element.addEventListener(event, function)`	Attaches an event listener to the element
`element.removeEventListener(event, function)`	Removes an event listener from the element

Creating a Simple Task List Application

Let's see some of the DOM manipulation methods in action by creating a simple task list application. The application has following features:

1. Add a task by providing title and description using a form
2. Dynamically generate the DOM element of task and append it to the DOM
3. Mark task as complete - remove the element from DOM

Create the HTML Markup

1. Create a file `index.html`
2. Add the following markup in it

```
<!DOCTYPE html>
<html>

<head>
    <title>DOM Manipulation</title>
</head>

<body>
    <section class="form-section">
        <form>
            <label for="title">Title:</label>
            <input type="text" id="title" name="title"><br>

            <label for="description">Description:</label>
            <input type="text" id="description" name="description"><br>

            <button type="button" onclick="createTask()">Create Task</button>
        </form>
    </section>
    <section>
        <div id="tasks">
        </div>
    </section>
    <script type="text/javascript" src="app.js"></script>
</body>

</html>
```

This is a basic HTML document that includes a form for creating tasks and a section for displaying the tasks that have been created.

The `head` section of the document contains a `title` element that sets the title of the page to "DOM Manipulation".

The `body` section of the document contains two `section` elements.

The first `section` element contains a `form` element that allows the user to input information about a task. The form includes two `label` elements and two `input` elements. The first `label` element has a `for` attribute that matches the `id` attribute of the first `input` element, and the second `label` element has a `for` attribute that matches the `id` attribute of the second `input` element. This allows the user to click on the label to activate the corresponding input field. The `form` also includes a `button` element that calls the `createTask()` function when clicked.

The second `section` element contains a `div` element with an `id` of "tasks". This `div` element will be used to display the tasks that are created by the user.

Finally, there is a `script` element that references an external JavaScript file called "app.js". This file is used to define the `createTask()` function that is called when the "Create Task" button is clicked.

Create a JS file

Create a file named `app.js` and add the following piece of code:

```
function createTask() {
    //Get input values from the form
    const titleInput = document.getElementById('title');
    const descriptionInput = document.getElementById('description');
    const taskObject = {
        title: titleInput.value,
        description: descriptionInput.value
    };

    //Get task container
    const taskContainer = document.getElementById('tasks');

    //Create a new task i.e. div element
    const taskDiv = document.createElement('div');
    taskDiv.className = "task";

    //Create a heading element and add it to the task div
    const headingElement = document.createElement("h3");
    headingElement.textContent = taskObject.title;
    taskDiv.appendChild(headingElement);

    //Create a paragraph element and add it to the task div
    const paragraphElement = document.createElement("p");
    paragraphElement.textContent = taskObject.description;
    taskDiv.appendChild(paragraphElement);

    //Create a button element and add it to the task div
    const buttonElement = document.createElement("button");
    buttonElement.textContent = "Mark as Complete";
    buttonElement.addEventListener("click", () => {
        taskDiv.remove();
    });

    taskDiv.appendChild(buttonElement);

    taskContainer.appendChild(taskDiv);

    titleInput.value = "";
    descriptionInput.value = "";

}
```

This is a JavaScript function called `createTask()` that creates a new task based on the user input and adds it to the page.

The function first gets the input values from the form's "title" and "description" input fields using the `getElementById()` method and stores them in a new `taskObject` object, which has two properties: `title` and `description`.

The function then gets the task container element with the id "tasks" using the `getElementById()` method and stores it in a variable named `taskContainer`.

A new `div` element is created to represent the task and given a class of "task" using the `createElement()` method and `.className`.

A `h3` element is then created to represent the task title, and its text content is set to the `title` property of the `taskObject`. The `h3` element is appended to the `div`.

A `p` element is also created to represent the task description, and its text content is set to the `description` property of the `taskObject`. The `p` element is appended to the `div`.

A `button` element is also created and added to the `div`. The text content of the `button` is set to "Mark as Complete", and an event listener is added to it using the `addEventListener()` method. The event listener is a function that removes the `div` element representing the task (i.e. `taskDiv`) from the page when the button is clicked.

The `div` element representing the task is then appended to the task container (`taskContainer`) using the `appendChild()` method.

Finally, the last two lines of code clear the input values of the "title" and "description" input fields, respectively, so that the user can enter a new task.