DOM 2: Dynamic Interfaces

Programming 2 @ EK

KIIL 2026

DOM 2: Dynamic Interfaces

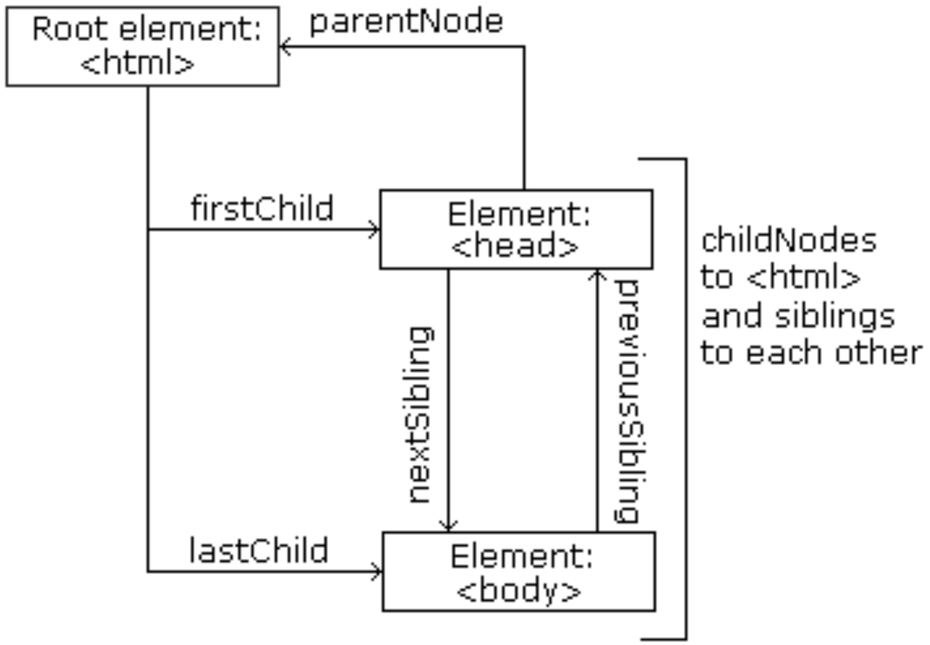
Agenda

- Recap on the DOM
- Event-driven Interactivity
- Dynamic UI Architecture

The Document Object Model (DOM)

Recap

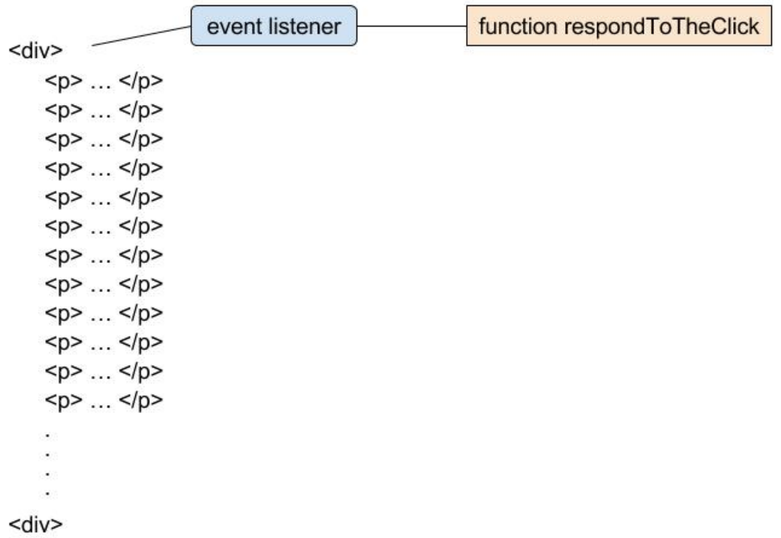
- Hierarchical Tree Structure
- A ‘bridge’ that turns every HTML tag into an object that Javascript can grab
- Without this API our webpage is static



Event-Driven Programming

Recap

- Making the UI react to asynchronous interactions
- Code is driven by external triggers (events)
- Browser utilises an ‘Event Loop’ to listen for user actions (clicks, inputs, etc.)



Reading State from the DOM

DOM as a temporary data store

- User input: Extracting dynamic, user-generated string values directly from interface elements (beware of type conversion!)
- HTML metadata: data-* attributes used to keep Javascript logic separated from CSS styling

The Element Lifecycle

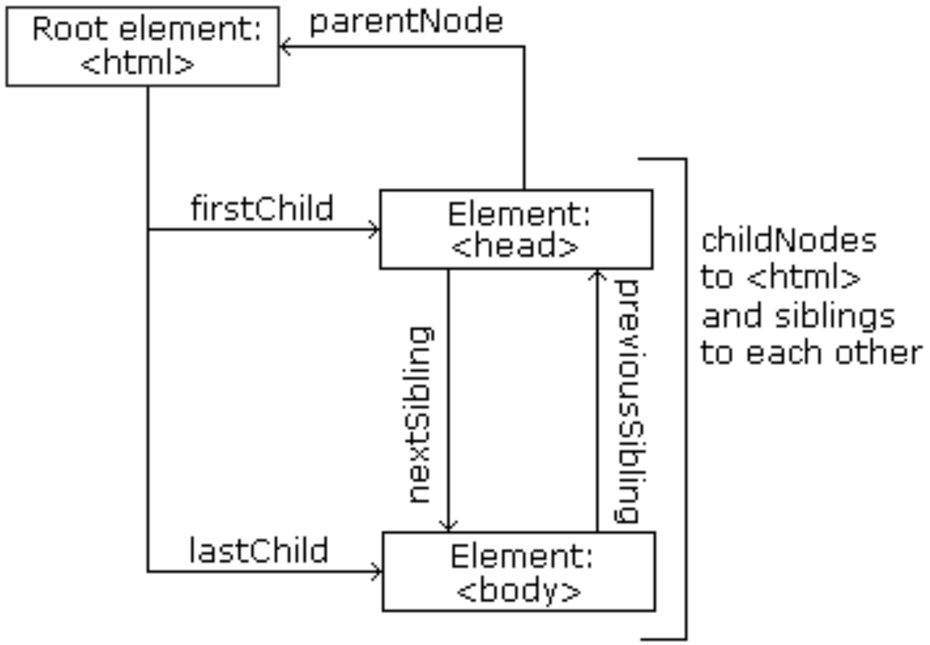
Standard pattern for generating dynamic UI

- Creation: Generate element object in memory (createElement)
- Configuration: Apply attributes, text, and event listeners to floating node
- Insertion: Trigger browser render by physically attaching node to DOM tree (appendChild, prepend, etc.)

Managing DOM Collections

Handling multiple interface components simultaneously

- Querying the DOM often returns a NodeList
- Managing collections require loops
- **‘Reset and Reapply’ Pattern:** Standard UI logic flow where the entire collection is reset to a default state before applying new state



Visual State Management

Updating the UI

- **Best Practice:** Javascript handles the logic, and CSS handles the design
- **Toggling Classes:** Instead of writing CSS styles directly inside Javascript, better practice is to add or remove class names (e.g. adding an ‘active’ class to a clicked tab)
- **Inline Styling:** Only use Javascript to apply direct styling when values are dynamic and hard to predict in CSS (e.g. mouse coordinates)